6-2020

A new framework for privacy-preserving biometric-based remote user authentication

Yangguang TIAN
Singapore Management University, ygtian@smu.edu.sg

Yingjiu LI

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

Nan LI

Pengfei WU

See next page for additional authors

Published in Journal of Computer Security, 2020, 28 (4), 469-498.
https://doi.org/10.3233/JCS-191336

A New Framework for Privacy-Preserving Biometric-Based Remote User Authentication

Yangguang Tian [a,*], Yingjiu Li [b], Robert H. Deng [a], Nan Li [c] Pengfei Wu [d], and Anyi Liu [e]

[a] *School of Information Systems, Singapore Management University, Singapore*
E-mails: sunshine.tian86@gmail.com, robertdeng@smu.edu.sg
[b] *Computer and Information Science, University of Oregon, USA*
E-mail: yingjiul@uoregon.edu
[c] *School of Electrical Engineering and Computing, University of Newcastle, NSW, Australia*
E-mail: nan.li@newcastle.edu.au
[d] *School of Software and Microelectronics, Peking University, Beijing, China*
E-mail: wpf9808@pku.edu.cn
[e] *Department of Computer Science and Engineering, Oakland University, USA*
E-mail: anyiliu@oakland.edu

Abstract. In this paper, we introduce the first general framework for strong privacy-preserving biometric-based remote user authentication based on oblivious RAM (ORAM) protocol and computational fuzzy extractors. We define formal security models for the general framework, and we prove that it can achieve user authenticity and strong privacy. In particular, the general framework ensures that: 1) a strong privacy and a log-linear time-complexity are achieved by using a new tree-based ORAM protocol; 2) a constant bandwidth cost is achieved by exploiting computational fuzzy extractors in the challenge-response phase of remote user authentications.

Keywords: Remote User Authentication, Oblivious RAM, Computational Fuzzy Extractors, Strong Privacy, Constant Bandwidth

1. Introduction

Privacy-Preserving Biometric-based Remote User Authentication (BRUA) allows an authorized user to anonymously authenticate herself to a remote authentication server using her biometrics. In the literature, BRUA with *biometrics privacy* has been intensively studied [1–6]. Biometrics privacy means that no biometrics should be stored in plaintext at server side, this is because a user may lose its security forever if her secret biometrics is leaked to the authentication server or any outsiders. However, we discover that none of existing schemes ever consider non-biometrics privacy, including *identity privacy* and *access privacy*.

*Corresponding author. E-mail: sunshine.tian86@gmail.com.

Identity privacy and access privacy are essential in the remote user authentication setting. A remote user authentication, that does not achieve identity privacy and access privacy, may leak sensitive information about a user to the authentication server or any outsiders. Let us consider a real-world application: mobile health (mHealth), where patients wish to obtain healthcare information after being authenticated by an authentication server through mobile devices. We assume an authentication server maintains a database of records and each record contains a patient's "encrypted" biometrics. We also assume that the authentication server must access a patient's record to authenticate the patient. Identity privacy can ensure that a patient logs in to mHealth system without disclosing her real identity to the authentication server.

We note that the previously mentioned login does not guarantee access privacy. Access privacy means that the authentication server cannot determine which record is being accessed at any time. If the same record is accessed twice, then the authentication server can easily link two accesses made by the same anonymous patient [7, 8]. As a result, the authentication server can learn a patient's sensitive information such as interaction history and login behaviour, and disclose it to third parties afterwards [9]. In contrast, access privacy aims to prevent the authentication server from obtaining such sensitive information.

The practicality of a remote user authentication system (e.g., mHealth) is evaluated by *time-complexity* and *bandwidth cost*. First, the time-complexity means that the amount of time it takes for the authentication server to authenticate a user. Second, the bandwidth cost means the number of records transferred between the authentication server and a user in order to authenticate the user. To analyze the bandwidth cost in detail, we assume a remote user authentication includes three phases: early-reshuffle, challenge-response and post-reshuffle. Both early-reshuffle and post-reshuffle allow a patient to reshuffle and re-randomize a set of records in the database, so multiple logins by the same anonymous patient are unlinkable by the authentication server. In the challenge-response phase, a patient proves her authenticity to the authentication server by generating digital signatures based on any messages (e.g., nonces) and transferred records.

In this work, we present the first general framework of biometrics-based remote user authentication that satisfies *strong privacy*, including biometrics privacy, identity privacy and access privacy, while *the time-complexity is log-linear in the number of enrolled users, and the communication bandwidth in the challenge-response phase is constant*. We call our framework pBRUA for convenience.

Overview of Techniques. We now explain our key technical insights. First, *biometrics privacy* and *identity privacy* can be achieved using the existing techniques such as computational fuzzy extractors [10–12] (note that fuzzy extractors are used to convert repeated noisy readings of a secret into the same key of uniform distribution [13]) and anonymous digital signatures [14, 15]. Second, for achieving *access privacy*, to the best of our knowledge, there is no existing solution that can be directly applied to construct pBRUA[1]. We shall show that some existing tree-based ORAM protocols can be used to construct BRUA with log-linear time-complexity (see Appendix A). However, they cannot achieve constant bandwidth cost in the challenge-response phase. Therefore, we propose a new tree-based ORAM (uORAM) protocol (see Figure 1). The uORAM not only supports a *log-linear time-complexity* in the number of records (including all enrolled users' real records and many dummy records) due to the structure of a binary tree, but also achieves a *constant bandwidth* in the challenge-response phase.

Our Novel Technique. In pBRUA, a user first reshuffles and re-randomizes a set of records in a tree path in the early-reshuffle phase, which includes a real record (which contains the user's "encrypted" biomet-

[1] It is possible to use other alternative solutions to achieve access privacy such as private information retrieval [16] and shuffle index [17], while the ORAM based solution may achieve lower bandwidth complexity.

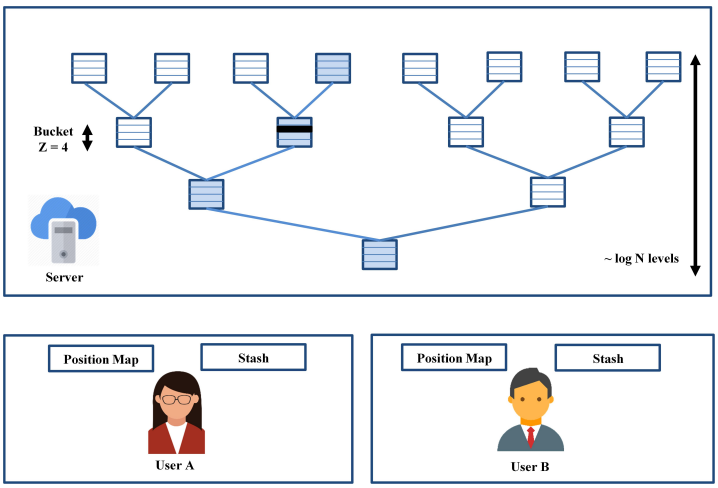

Fig. 1. uORAM in a multiple-user and single-server setting. The individual user stores a small amount of local data in a stash. The server-side storage is treated as a binary tree where each node is a bucket that can hold up to a fixed number of records. If the black record is mapped to the shaded path, then the record must reside in any slot along the path or in the stash.

rics) and many dummy records (which contains "encrypted" zeros). The user is required to replace the "encrypted" zeros in the dummy records with secret randomness chosen by the user. In the challenge-response phase, the authentication server "aggregates" the records in the tree path specified by the user and sends the "aggregated" record to the user. Then, the user obtains a cryptographic key from the "aggregated" record using her biometrics and the chosen secret randomness. Such cryptographic key is used to generate a signing/verification key pair to prove her authenticity. In particular, we use the Schnorr signature scheme [18]. This is because the cryptographic key is re-randomized by the user in the early-reshuffles phase. As a result, the Schnorr signature generated by the derived signing key in the challenge-response phase can achieve both user authenticity and constant bandwidth. We stress that we do not use the previously mentioned anonymous digital signatures, which essentially compromise the constant bandwidth, because the number of verification keys transferred between user and server for verifying anonymous signatures has the log-linear time-complexity in the number of records.

Our Contributions. The major contributions of this work are summarized as follows.

- *General Framework.* We propose the first general framework pBRUA using learning with errors (LWE) computational fuzzy extractors [10–12], digital signatures [19, 20] and a uORAM protocol. The proposed pBRUA achieves the strong privacy, log-linear time-complexity and constant bandwidth in the challenge-response phase. We prove that the proposed pBRUA can achieve user authenticity and strong privacy under standard assumptions.
- *New ORAM.* We propose a new tree-based ORAM (uORAM) for remote user authentications in a multi-user setting, which is built on top of Path ORAM [21] and Ring ORAM [22] (a variant of Path ORAM). The proposed uORAM is proven secure under a variant of standard ORAM security definition [23].
- *Constant Bandwidth.* We show the proposed uORAM (and pBRUA as well) can achieve the constant bandwidth. Constant bandwidth in this work means that the authentication server transfers a *single* record to an authorized user in the challenge-response phase.

Table 1

N is the total number of records, Z (e.g., $Z = 4$) is the number of real records per bucket (more details are referred to Section 3) and Z^* (e.g., $Z^* = 3$) is a smaller number than Z which means an improved overall bandwidth in Ring ORAM.

Criteria/Construction	Path	Ring	uORAM	uORAM + **FE**
Overall Bandwidth	$2Z \cdot \log N$	$Z^* \cdot \log N$	$2Z \cdot \log N$	$2Z \cdot \log N$
CR Bandwidth	$Z \cdot \log N$	$\log N$	$\log N$	**1**

We highlight our contributions in Table 1. We can see that *the pBRUA is achieved by proposing a uORAM and exploiting the LWE-based fuzzy extractor* **(FE)** *altogether*. We remark that the new uORAM protocol can be regarded as an independent cryptographic primitive, if one does not consider the constant bandwidth in the challenge-response (**CR**) phase.

1.1. Related Work

ORAM. Oblivious RAM was introduced by Goldreich and Ostrovsky [23] (GO-ORAM), that allows a client to conceal her access pattern as seen by the untrusted storage server. They have proposed two ORAMs: Square-root ORAM and Hierarchical ORAM. The main drawback is: the worst-case cost on bandwidth is linear in the total number of records (or blocks) N. The bandwidth is to measure the amount of communication cost between client and server to serve a client request. For example, the Hierarchical ORAM is with $\mathcal{O}(N \cdot \log^2 N)$ (i.e., poly-logarithmic) complexity, so it hinders its practicality in realistic settings. To reduce the worst-case cost, Shi et al. [24] proposed the tree-based ORAM which manages the ORAM storage into a binary tree, so that achieving the worst-case bandwidth with $\mathcal{O}(\log^3 N)$ complexity.

To further reduce the bandwidth cost while keeping a small client storage, Stefanov et al [21] proposed the Path ORAM with $\mathcal{O}(\log N)$ bandwidth complexity. The Path ORAM is extremely simple—just 16 lines of pseudocode. Each access can be expressed a fetching and storing a path in the tree stored remotely on the server. However, the overall bandwidth is too high because the server has to pass all blocks in a tree path to client, and the overall bandwidth of Path ORAM depends on the bucket size. To remove such dependence, Ren et al. proposed the Ring ORAM [22] such that fetching one block per bucket in a tree path. Moreover, the Ring ORAM can achieve the $\mathcal{O}(1)$ on-line bandwidth efficiency by applying the XOR technique [25], which means that returning a single block back to serve a user request. This is important because the on-line bandwidth determines the response time to serve a user request. We note that the **Evict** operation (see Section 3) in the Ring ORAM is NOT suitable for user authentications because it does not execute on every user request. By contrast, the requested user should push blocks back to ORAM tree after a valid authentication because multiple users share the same ORAM tree.

For achieving the $\mathcal{O}(1)$ on-line bandwidth efficiency, the untrusted server is allowed to perform matrix multiplication on some blocks (e.g., SSS ORAM [26]) or XOR operation (e.g., Burst ORAM [25] and Ring ORAM [22]). In particular, the Onion ORAM [27] can achieve a constant worst-case bandwidth blowup by allowing the untrusted server to perform the (additive/somewhat) homomorphic encryption on the involved blocks. The bandwidth blowup means the number of blocks transmitted between client and server to serve a client request. For example, Circuit ORAM [28] incurs a $\mathcal{O}(\log N)$ lower bound in bandwidth blowup, while Onion ORAM has a $\mathcal{O}(1)$ bandwidth blowup.

A separate line of research on ORAM is to handle the asynchronous user requests at multi-user setting [29, 30], Sahin et al. [31] introduced a new ORAM: Taostore, which relies on a trusted proxy who acts as a middle layer between multiple users and an untrusted server (i.e., "hybrid cloud" model [29]).

Meanwhile, many practical ORAMs have been implemented for real-world applications, like secure processors [21, 32], secure storage systems [25, 26, 29] and secure multi-party computations [28, 33].

Comparison with Existing ORAMs. First, we notice that some existing tree-based ORAMs can be directly used to construct BRUA with log-linear time-complexity (e.g., the Path ORAM described in Appendix A). Then, we compare our uORAM with some existing tree-based ORAMs in Table 2. Our uORAM protocol is unique in design: 1) the authentication server generates certain number of dummy records for padding each bucket during enrollment, in which the dummy records are common resources shared by all enrolled users; 2) an enrolled user maintains her access privacy during authentication, which is usually executed in a short time period; 3) multiple users store their individual key materials securely and maintain stash independently. Our designed uORAM works at non-standard model, secures against an honest-but-curious (or semi-honest) server. It has two round trips in total, and a single server coordinates the authentication requests from multiple users in sequence. We note that the proposed uORAM can store a generic data as required in the existing ORAMs. In this work, we use the LWE-based fuzzy extractor to store biometrics with a specific format, so as to achieve a constant bandwidth for user authentication. If the constant bandwidth is not required in the design goal, then many fuzzy extractors in the literatue could be suitable for uORAM which can handle various types of biometrics.

Table 2

The overall comparison between our uORAM and some tree-based ORAMs. Bandwidth means on-line bandwidth complexity. Standard means no server computation, non-standard means server can perform certain computation such as XOR. "1" round trip means the $\mathcal{O}(\log N)$ bandwidth complexity when processing the buckets in parallel. Multi-User means that whether a single ORAM contains the blocks from multiple clients. Asynchronicity means whether server handles multiple client requests asynchronously. Security means whether the untrusted server is semi-honest (SH) or malicious (M). In Ring [22], it achieves $\mathcal{O}(1)$ on-line bandwidth complexity using the XOR technique. In Taostore [31], the total number of round-trips depends on the number of asynchronous client requests (here we denote it as N/A). In uORAM, the on-line bandwidth complexity $\mathcal{O}(1)$ covers the challenge-response phase of user authentications.

Criteria/ORAM	Tree [24]	Path [21]	Ring [22]	Onion [27]	Taostore [31]	**Ours**
Bandwidth	$\mathcal{O}(\log^3 N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\log N)$	$\mathcal{O}(1)$
Standard	✓	✓	×	×	✓	×
Roundtrips	1	1	2	1	N/A	2
Multi-User	×	×	×	×	✓	✓
Asynchronicity	×	×	×	×	✓	×
Security	N/A	N/A	SH	SH/M	SH	SH

Fuzzy Extractor. Fuzzy extractor (FE) is one of the promising approaches to construct a biometric-based user authentication [1, 2, 6]. Juels and Wattenberg [34] introduced a cryptography primitive called "fuzzy commitment". It can be used in the biometric-based authentication systems, because its error-correcting technique can correct certain errors within a suitable metric (e.g., Hamming distance). Dodis et al. [13] formally introduced the notions of "secure sketches" (the sketches are used to recover the original biometrics from a nearby biometrics) and "fuzzy extractors". In particular, they provided concrete constructions of secure sketches and FEs in three metrics (Hamming distance, set difference and edit distance), and the constructions are information-theoretically secure.

Fuller et al. [10] introduced the first computationally-secure FE from LWE [35] such that the derived cryptographic key equals to the entropy[2] of the fuzzy biometrics. However, their computational FE is

[2]To obtain sufficient entropy at one time, a sensor that captures multiple biometrics (e.g., fingerprint and fingervein) has been developed [36], but we do not survey on this research direction since it is beyond the scope of this work.

not reusable. Reusability [1] means that a user can produce multiple key and sketch pairs using the same biometrics w (i.e., $\{(s_i, p_i)\} \leftarrow \mathsf{Gen}(w)$). To achieve a reusable FE from LWE, Apon et al. [11] provided a general method to convert non-reusable (resp. weakly reusable) computational fuzzy extractors to weakly reusable (resp. strongly reusable) ones. The strongly reusable FE allows an attacker to obtain the secret key string s_i, in addition to the public sketch p_i. We notice that the weakly reusable FE will suffice for privacy-preserving remote user authentications in this work, and we can achieve the strongly reusability by using the general method proposed in [11] when considering multiple authentication servers. In addition, we present the commonly used notations in Table 3.

Table 3
Summary of notations

Notation	Meaning
n/N	Total number of users/records (blocks)
$L = \log N$	Height of the binary tree $\mathcal{T}$
$ID/leaf_{ID}$	Block/leaf identifier
$Z/S/\mathsf{S}$	Real block/dummy block/Stash
$\mathtt{pk}_i/\mathtt{sk}_i$	User i public/secret key pair
$\mathsf{dist}(x, y)$	Distance between vector x and vector y
$t \in \mathbb{R}^+$	Threshold value (positive real number)
w/w'	Enrolled biometrics/noised biometrics
$\mathsf{SS}(s, w)$	Secure sketch with a secret string s
$\mathcal{P}(leaf_{ID})$	Tree path from leaf node $leaf_{ID}$ to the root
$\mathcal{P}(leaf_{ID}, \ell)$	The bucket at level ℓ along the tree path $\mathcal{P}(leaf_{ID})$
$position$	User's local position map
$leaf_{ID} = position(ID)$	Block ID resides somewhere along the path $\mathcal{P}(leaf_{ID})$

1.2. Paper Organization

In the next section, we present some preliminaries which will be used in our proposed construction. In Section 3, we present our uORAM protocol and its security analysis. We then present our general framework of pBRUA in Section 4 and formally prove its security in Section 5. The paper is concluded in Section 6.

2. Preliminaries

In this section, we present the digital signatures with homomorphic property, a family of universal hash functions and fuzzy extractors from LWE, which will be used in our proposed general framework.

2.1. Complexity Assumptions

Definition 2.1 (Decisional LWE [35])**.** *Given a random matrix $\mathbb{A} \in \mathbb{Z}_q^{m_0 \times n_0}$, $X \in \mathbb{Z}_q^{n_0}$ and χ be an arbitrary distribution on $\mathbb{Z}_q^{m_0}$, the decisional LWE (*D-LWE$_{q,n_0,m_0,\chi}$*) problem is to distinguish the distribution $(\mathbb{A}, \mathbb{A} \cdot X + \chi)$ from a random distribution over $(\mathbb{Z}_q^{m_0 \times n_0}, \mathbb{Z}_q^{m_0})$. We say that* D-LWE$_{q,n_0,m_0,\chi}$ *is (ϵ, s_{sec}) secure if no PPT distinguisher $\mathcal{D}_{s_{sec}}$ of size s_{sec} can distinguish the LWE instances from uniform except with probability ϵ, where $s_{sec} = poly(\lambda)$ and ϵ is a negligible function of the security parameter λ.*

Dottling and Muller-Quade [37] showed that one can encode biometrics w as the error term in a LWE problem by splitting it into m_0 blocks. Furthermore, to extract the pseudorandom bits, we rely on the result from Akavia et al. [38] such that $X \in \mathbb{Z}_q^{n_0}$ has simultaneously many hardcore bits.

Lemma 2.1. *If* D-LWE$_{q,n_0-k,m_0,\chi}$ *is* (ϵ, s_{sec}) *secure, then*

$$\delta^{\mathcal{D}_{s'_{sec}}}((X_{1,\cdots,k}, \mathbb{A}, \mathbb{A}\cdot X+\chi), (U, \mathbb{A}, \mathbb{A}\cdot X+\chi)) \leqslant \epsilon,$$

where $U \in \mathbb{Z}_q^k$ *and* $X_{1,\cdots,k}$ *denotes the first k coordinates of x and* $s'_{sec} \approx s_{sec} - n_0^3$.

2.2. Digital Signatures

We say a digital signature scheme $\Sigma = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Sign}, \mathsf{Verify})$ is homomorphic, if the following conditions are held.

(1) *Simple Key Generation.* $(\mathtt{sk}, \mathtt{pk}) \leftarrow \Sigma.\mathsf{KG}(pp)$ and $pp \leftarrow \Sigma.\mathsf{Setup}(\lambda)$, where $\mathtt{pk}$ is derived from $\mathtt{sk}$ via a deterministic algorithm $\mathtt{pk} \leftarrow \mathsf{KG}'(pp, \mathtt{sk})$.

(2) *Linearity of Keys.* $\mathsf{KG}'(pp, \mathtt{sk} + \Delta(\mathtt{sk})) = M_{\mathtt{pk}}(pp, \mathsf{KG}'(pp, \mathtt{sk}), \Delta(\mathtt{sk}))$, where $M_{\mathtt{pk}}$ denotes a deterministic algorithm which takes pp, a public key $\mathtt{pk}$ and a "shifted" value $\Delta(\mathtt{sk})$, outputs the "shifted" public key $\mathtt{pk}'$.

(3) *Linearity of Signatures.* Two distributions are identical: $\{\sigma' \leftarrow \Sigma.\mathsf{Sign}(pp, \mathtt{sk} + \Delta(\mathtt{sk}), m)\}$ and $\{\sigma' \leftarrow M_\Sigma(pp, \mathtt{pk}, m, \sigma, \Delta(\mathtt{sk}))\}$, where $\sigma \leftarrow \Sigma.\mathsf{Sign}(pp, \mathtt{sk}, m)$ and M_Σ denotes a deterministic algorithm which takes pp, a public key $\mathtt{pk}$, a message-signature pair (m, σ) and a "shifted" value $\Delta(\mathtt{sk})$, outputs the "shifted" signature σ'.

(4) *Linearity of Verifications.* We require that $\Sigma.\mathsf{Verify}(pp, M_{\mathtt{pk}}(pp, \mathtt{pk}, \Delta(\mathtt{sk})), m, M_\Sigma(pp, \mathtt{pk}, m, \sigma, \Delta(\mathtt{sk}))) =$ "1", and $\Sigma.\mathsf{Verify}(pp, \mathtt{pk}, m, \sigma) =$ "1".

Matsuda et al [20] showed that the Schnorr signature scheme [18] satisfies the homomorphic properties regarding keys and signatures (see Lemma 2 in [20]).

2.3. Universal Hash Function

Let $\mathbb{H}$ be a universal hash function family whose domain is $\mathbb{Z}_{q^{n_0}}$ and whose range is $\mathbb{Z}_q$. Let $\mathbb{Z}_{q^{n_0}}$ be a vector space, which consists of n_0 dimensional of finite ring with prime order q. We define an isomorphism $\psi : (\mathbb{Z}_q)^{n_0} \to \mathbb{Z}_{q^{n_0}}$ (ψ^{-1} is its inverse), and $n_0 \in \mathbb{N}$. Note that $(\mathbb{Z}_q)^{n_0} = \mathbb{Z}_q^{n_0}$. A family of universal hash functions is defined as $\mathbb{H} = \{\mathtt{H}_z : \mathbb{Z}_q^{n_0} \to \mathbb{Z}_q | z \in \mathbb{Z}_{q^{n_0}}\}$. Specifically, for each invertible element $z \in Z$ in the seed space $Z \in \mathbb{Z}_{q^{n_0}}$, define the hash function $\mathtt{H}_z$ as follows: on input $x \in (\mathbb{Z}_q)^{n_0}$, $\mathtt{H}_z(x)$ computes $y \leftarrow \psi(x) \cdot z$, where " $\cdot$ " denotes the multiplication in the extension field $\mathbb{Z}_{q^{n_0}}$. Let $(y_1, \cdots, y_{n_0}) \leftarrow \psi^{-1}(y)$, and the output of $\mathtt{H}_z(x)$ is $y_1 \in \mathbb{Z}_q$. Since the isomorphism ψ between $\mathbb{Z}_q^{n_0}$ and $\mathbb{Z}_q^{n_0}$ is applied to the universal hash function family, we can easily get the the desired linearity below

$$\forall x, x' \in (\mathbb{Z}_q)^{n_0} \text{ and } y_1, y_2 \in \mathbb{Z}_q : y_1 \cdot \mathtt{H}_z(x) + y_2 \cdot \mathtt{H}_z(x') = \mathtt{H}_z(y_1 \cdot x + y_2 \cdot x').$$

Lemma 2.2. *Assume a family of functions* $\{\mathtt{H}_z : \mathbb{Z}_q^{n_0} \to \mathbb{Z}_q\}_{z\in Z}$ *is universal, for any random variable W taking values in* $\mathbb{Z}_q^{n_0}$ *and any random variable Y,*

$$\mathbf{SD}((U_Z, \underline{\mathtt{H}_z(W)}, Y), (U_Z, \underline{U}, Y)) \leqslant \frac{1}{2}\sqrt{2^{-\tilde{\mathbf{H}}_\infty(W|Y)} \cdot |\mathbb{Z}_q|},$$

where U_Z *and* U *are uniformly distributed over* $\mathbb{Z}_{q^{n_0}}$ *and* $\mathbb{Z}_q$ *respectively. In particular, such universal hash functions are (average-case, strong) extractors with* ϵ*-statistically close to uniform. The detailed description of (average) mini-entropy* $\tilde{\mathbf{H}}_\infty$ *and statistical distance* **SD** *can be found in [13].*

2.4. Fuzzy Extractors

Let $m_0 \geqslant n_0$ and q be a prime number, define two algorithms Gen, Rep of computational FE [10] below.

(1) Input: $w \leftarrow \mathcal{M}$. $\triangleright$ suppose $\mathcal{M}$ is a uniform distribution over $\mathbb{Z}_q^{m_0}$.
(2) Sample $\mathbb{A} \in \mathbb{Z}_q^{m_0 \times n_0}$, $x \in \mathbb{Z}_q^{n_0}$ uniformly.
(3) Compute $p = (\mathbb{A}, \mathbb{A} \cdot x + w)$, $s = x_{1,\cdots,n_0/2}$.
(4) Output: (s, p).

(1) Input: w', p. $\triangleright$ where Hamming distance between w' and w is at most t.
(2) Parse p as $(\mathbb{A}, c)$; let $b = c - w'$.
(3) Let $x = \mathsf{Decode}_t(\mathbb{A}, b)$.
(4) Output: $s = x_{1,\cdots,n_0/2}$.

Note that p denotes the public helper string, and s denotes the secret string. The correctness of computational FE relies on the $\mathsf{Decode}_t(\mathbb{A}, b)$ algorithm [10], which is explicitly shown as follows.

(1) Input: $(\mathbb{A}, b = \mathbb{A} \cdot x + w - w')$.
(2) Select $2n_0$ distinct indices $i_1, \cdots, i_{2n_0} \leftarrow \{1, \cdots, m_0\}$.
(3) Restrict $\mathbb{A}, b$ to rows $i_1, \cdots, i_{2n_0}$; Denote these by $\mathbb{A}_{i_1}, \cdots, \mathbb{A}_{i_{2n_0}}, b_1, \cdots, b_{i_{2n_0}}$.
(4) Find n_0 linearly independent rows of $\mathbb{A}_{i_1}, \cdots, \mathbb{A}_{i_{2n_0}}$ (if no such rows exist, output abort and stop), and restrict $\mathbb{A}_{i_1}, \cdots, \mathbb{A}_{i_{2n_0}}, b_{i_1}, \cdots, b_{i_{2n_0}}$ to n_0 rows. Denote the result by $\mathbb{A}', b'$.
(5) Compute $x' = \mathbb{A}'^{-1} \cdot b'$.
(6) If $b - \mathbb{A} \cdot x'$ has at most t non-zero coordinates, then outputs x'; Otherwise, it returns to step 2.

Recall that $\mathbb{A} \in \mathbb{Z}_q^{m_0 \times n_0}, b \in \mathbb{Z}_q^{m_0}$, and Decode_t algorithm can correct at most $t = \mathcal{O}(\log n_0)$ errors (of Hamming distance) in a random linear code. Also note that with probability at least $1/poly(\lambda)$, none of the $2n_0$ rows selected in step 2 have errors (i.e., nearby biometrics w and w' agree on these rows), thus x' is a solution to the linear system. Furthermore, we notice that the sketch from LWE satisfies the linearity defined in [19, 20]. That is,

$$\mathsf{SS}(w, x + \Delta(x)) = \mathbb{A} \cdot (x + \Delta(x)) + w = (\mathbb{A} \cdot x + w) + \mathbb{A} \cdot \Delta(x),$$

where SS denotes a secure sketch procedure [10], which takes $w \leftarrow \mathcal{M}$ and a value $x \in \mathbb{Z}_q^{n_0}$ as input, output a distribution over $\mathbb{Z}_q^{m_0}$. The sketch from LWE is in the form of $\mathsf{SS}(w, x) = \mathbb{A} \cdot x + w$.

The computational fuzzy extractors (FE) from LWE has an inherent property: “indistinguishability". Informally, given two sketches (part of helper data) with respect to two independent biometrics, adversary cannot distinguish them without having decryption keys. We formally prove that the computational FE from LWE is secure in the IND model (note that the adversary here is allowed to access the sketch only). In addition, we discover that both computational FE [10] and its variant reusable FEs [11, 12] have such inherent property.

Definition 2.2. *The IND experiment between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ is defined below.*

Experiment $\mathsf{Exp}_{FE}^{IND}(\lambda)$
$b \in \{0,1\}, w_b \leftarrow \mathcal{M}, \mathcal{Q} = \emptyset$
  $(r_i, p_i) \leftarrow \mathsf{Gen}(w_b)$
  $\mathcal{Q} \leftarrow \mathcal{Q} \cup p_i$
  return p_i
$b' = \mathcal{A}(guess, c^*), c^* \leftarrow p_b$
Return $1, if\ b' = b \wedge p_b \notin \mathcal{Q}; else, return\ 0.$

In the guess stage, $\mathcal{A}$ is given a challenge sketch c^*, which was not previously simulated by $\mathcal{C}$. We define the advantage of $\mathcal{A}$ as

$$\mathtt{Adv}_{\mathcal{A}}^{IND}(\lambda) = |\Pr[\mathcal{C} \rightarrow 1] - 1/2|.$$

A computational FE from LWE is said to be IND secure if $\mathtt{Adv}_{\mathcal{A}}^{IND}(\lambda)$ is negligible in λ for any PPT $\mathcal{A}$.

Lemma 2.3. *The computational fuzzy extractors from LWE achieves the IND security if the* D-LWE$_{q,n_0,m_0,\chi}$ *assumption is (ϵ, s_{sec}) secure.*

Informally, we can think of the sketch $\mathbb{A} \cdot x + w$ (part of helper data p) as an “encryption" of x that where decryption works from any close w' (i.e., decryption key). Furthermore, we can also think of any two sketches $\mathbb{A}_0 \cdot x_0 + w_0$ and $\mathbb{A}_1 \cdot x_1 + w_1$ (in a multi-user setting, we set $\mathbb{A}_0 = \mathbb{A}_1$ which is shared among all users) are indistinguishable by any third parties without having decryption keys (w_0', w_1').

Adversary $\mathcal{C}(\mathbb{A}, v)$
$b \in \{0,1\}, u \xleftarrow{R} \mathbb{Z}_q^{n_0}, w \xleftarrow{R} \mathbb{Z}_q^{m_0}$
  $p_0 \leftarrow \mathbb{A} \cdot X + \chi + w_0; p_1 \leftarrow \mathbb{A} \cdot (X + u) + \chi + w_1$
$b' = \mathcal{A}(\text{guess}, c^*), c^* \leftarrow p_b$
$If\ b' = b, return\ 1; else, return\ 0.$

Fig. 2. Description of adversary $\mathcal{C}$ for the proof

Proof. Assume that there exists a PPT $\mathcal{A}$ breaking the IND security of the computational fuzzy extractors from LWE, then we can construct an algorithm $\mathcal{C}$ to break the decisional LWE (*D-LWE*$_{q,n_0,m_0,\chi}$)

assumption. The algorithm $\mathcal{C}$ has almost the same time complexity with $\mathcal{A}$. For simplicity, we consider the shared public parameter by all users such that $\mathbb{A}_0 = \mathbb{A}_1$.

The algorithm $\mathcal{C}$ uses $\mathcal{A}$ as a subroutine (see Fig. 2, note that v can be either $\mathbb{A} \cdot X + \chi$ or a random distribution). $\mathcal{C}$ first generates another distribution which has the same property and distribution as its own challenge distribution. That is, computed distribution $(\mathbb{A}, \mathbb{A} \cdot (X+u) + \chi + w)$, where u and w are randomly chosen by $\mathcal{C}$. If $\mathcal{C}$'s challenge is a real distribution, then it is the computed distribution; Otherwise, it is a random distribution over $(\mathbb{Z}_q^{m_0 \times n_0}, \mathbb{Z}_q^{m_0})$. By using its challenge and computed distribution, $\mathcal{C}$ can simulate two sketches (p_0, p_1) for $\mathcal{A}$. At guess stage, $\mathcal{C}$ returns a challenge ciphertext c^* to $\mathcal{A}$ according to the bit b.

We then analyze the behaviour of $\mathcal{C}$ on $\mathsf{Exp}_{\mathcal{C}}^{\text{LWE-REAL}}$ and $\mathsf{Exp}_{\mathcal{C}}^{\text{LWE-RAND}}$ respectively. In the $\mathsf{Exp}_{\mathcal{C}}^{\text{LWE-REAL}}$, the input $(\mathbb{A}, \mathbb{A} \cdot X + \chi + w)$ satisfies the Rep algorithm of FE described in Section 2.4. Notice that the computed distribution $(\mathbb{A}, \mathbb{A} \cdot (X+u) + \chi + w)$ is also valid and they are uniformly and independently distributed over $(\mathbb{Z}_q^{m_0 \times n_0}, \mathbb{Z}_q^m)$, because $\mathbb{A} \cdot (X+u) + \chi + w = \mathbb{A} \cdot X + \chi + \mathbb{A} \cdot u + w$ and u is a randomly element in $\mathbb{Z}_q^{n_0}$. Thus, $\mathcal{C}$ can simulate the proper distribution of two challenge sketches (i.e., $p_0 \leftarrow \mathbb{A} \cdot X + \chi + w_0$ and $p_1 \leftarrow \mathbb{A} \cdot (X+u) + \chi + w_1$), and the challenge ciphertext c^* is distributed exactly like a real sketch which associates with w_b.

$$c_0 \leftarrow \mathbb{A} \cdot X + \chi + w_0. \triangleright \ if\ b = 0$$
$$c_1 \leftarrow \mathbb{A} \cdot (X+u) + \chi + w_1. \triangleright \ otherwise$$

Therefore, we have $\mathsf{Exp}_{\mathcal{C}}^{\text{LWE-REAL}}$ below, which includes the experiment with respect to $b = 1$ (i.e., IND-1) and $b = 0$ (i.e., IND-0).

$$\begin{aligned}\Pr[\mathsf{Exp}_{\mathcal{C}}^{\text{LWE-REAL}}(\lambda) = 1] &= 1/2 \cdot \Pr[\mathsf{Exp}_{\mathcal{A}}^{\text{IND-1}}(\lambda) = 1] + 1/2 \cdot (1 - \Pr[\mathsf{Exp}_{\mathcal{A}}^{\text{IND-1}}(\lambda) = 1]) \\ &= 1/2 + 1/2 \cdot \mathtt{Adv}_{\mathcal{A}}^{\text{IND}}(\lambda).\end{aligned}$$

As for $\mathsf{Exp}_{\mathcal{C}}^{\text{LWE-RAND}}$, the input distributions to $\mathcal{C}$ in Fig. 2.4 are all uniformly distributed over $(\mathbb{Z}_q^{m_0 \times n_0}, \mathbb{Z}_q^{m_0})$. Therefore, the corresponding computed distribution above are also uniformly and independently distributed over $(\mathbb{Z}_q^{m_0 \times n_0}, \mathbb{Z}_q^{m_0})$. In particular, the challenge ciphertext is a random distribution over $(\mathbb{Z}_q^{m_0 \times n_0}, \mathbb{Z}_q^{m_0})$, and independent of bit b. Hence we have

$$\Pr[\mathsf{Exp}_{\mathcal{C}}^{\text{LWE-RAND}}(\lambda) = 1] \leqslant 1/2 + 1/2^{\lambda-1}.$$

The last term indicates that the random distribution to $\mathcal{C}$ happen to have the distribution of a real distribution, which is bounded by $1/2^{\lambda-1}$ since $2^{\lambda-1} < q < 2^{\lambda}$. By combing all equations above, we have

$$\begin{aligned}\mathtt{Adv}_{\mathcal{C}}^{\text{LWE}}(\lambda) &= \Pr[\mathsf{Exp}_{\mathcal{C}}^{\text{LWE-REAL}}(\lambda) = 1] + \Pr[\mathsf{Exp}_{\mathcal{C}}^{\text{LWE-RAND}}(\lambda) = 1] \\ &\geqslant 1/2 \cdot \Pr[\mathsf{Exp}_{\mathcal{A}}^{\text{IND}}(\lambda) - 1/2^{\lambda-1}.\end{aligned}$$

We can also show that $\mathbb{A}_0 \neq \mathbb{A}_1$ when public parameter is chosen at random over $\mathbb{Z}_q^{m_0 \times n_0}$, while $\mathcal{C}$ will slightly change to $p_0 \leftarrow (\mathbb{A}_0 = \mathbb{A}, \mathbb{A}_0 \cdot X + \chi + w_0); p_1 \leftarrow (\mathbb{A}_1 = \mathbb{A}_0 \cdot \mathbb{A}^*, \mathbb{A}_1 \cdot (X+u) + \chi + w_1)$, where $\mathbb{A}^* \xleftarrow{R} \mathbb{Z}_q^{m_0 \times n_0}$.

□

3. A New Oblivious RAM

In this section, we present our proposed tree-based ORAM (uORAM). The uORAM protocol is comprised of some novel techniques from Path ORAM [21] and Ring ORAM [22].

Binary Tree $\mathcal{T}$. We assume a binary tree of height $L = \log N$ and 2^L leaves. Levels in the tree are numbered from 0 (the root) to L.

Bucket. Each node in the tree is called a bucket with a fixed number (e.g., 4 or 5) of blocks. We guarantee that each bucket has exactly Z real blocks, and we allow the server to pad real blocks if a bucket has less than Z real blocks. We assume S dummy blocks are generated and padded for each bucket by the server. If the client tries to access her real blocks and maintain her access privacy, then she needs to update at least one dummy block in each bucket, which is similar to Path or Ring ORAM. Consequently, the server does not learn any information about the accessed blocks.

Path. Let $leaf_{ID} \in \{0, 1, \cdots, 2^L - 1\}$ denote the leaf node in the tree. We define $\mathcal{P}(leaf_{ID})$ as a set of buckets along the path from leaf $leaf_{ID}$ to the root, and $\mathcal{P}(leaf_{ID}, \ell)$ denotes the bucket in $\mathcal{P}(leaf_{ID})$ at level ℓ in $\mathcal{T}$.

Stash. During the course of the data access, a small number of blocks might overflow from the tree buckets on the server. The client can locally store these overflowing blocks in a local data structure S called stash.

Position Map. The client also locally maintains a position map (see Figure 1), which stores a public leaf identifier $leaf_{ID}$ and a secret block identifier ID such that $leaf_{ID} \leftarrow position[ID]$ (i.e., a block ID is currently mapped to a leaf node with identifier $leaf_{ID}$). The block ID either resides in some buckets in path $\mathcal{P}(leaf_{ID})$, or in the stash S. The position map changes over time as blocks are accessed and remapped.

Operations. uORAM includes the following key operations: **EarlyReshuffles**, **ReadPath** and **Evict**. Here we provide a high-level of these operations.

- **ReadPath (from Ring ORAM).** This operation reads one block from each bucket — the block of interest if found or a dummy block otherwise. Specifically, the **ReadPath** operation is to select a single block to read from that bucket. The index of the block to read (either real or random) is returned by the $\mathsf{GetBlockOffset}$ function. According to the random offset $offset$ per bucket and the leaf identifier $leaf_{ID}$, the server fetches these blocks and passes them to the client. Specifically, the client relies on a small-size **Bucket Metadata**, which is used to store the $offset$ per bucket. We stress that the $offset$ is chosen at random by client. The client must read through all the buckets in a tree path $\ell \in \{0, 1, \cdots, L\}$, and each bucket returns either the interested block or a randomly-chosen dummy block.

  (1) **ReadPath**$(leaf_{ID}, ID)$
  (2) $data \leftarrow \perp$, for $\ell \in \{0, 1, \cdots, L\}$:
    (a) $offset \leftarrow \mathsf{GetBlockOffset}(\mathcal{P}(leaf_{ID}, \ell), ID)$
    (b) $data' \leftarrow \mathcal{P}(leaf_{ID}, \ell, offset)$

(c) if $data' \neq \perp$ then $data \leftarrow data'$, return $data$.

- **EarlyReshuffles (from Ring ORAM).** To ensure the **ReadPath** operation securely read one block per bucket, it requires a maintenance task called **EarlyReshuffles** on $\mathcal{P}(leaf_{ID})$, the path accessed by **ReadPath**. That is, the client reshuffles the bucket in the path $leaf_{ID}$.

  (1) **EarlyReshuffles**$(leaf_{ID})$
  (2) for $\ell \in \{0, 1, \cdots, L\}$:
    (a) $\mathsf{S} \leftarrow \mathsf{S} \bigcup \mathsf{ReadBucket}(\mathcal{P}(leaf_{ID}, \ell))$
    (b) $\mathsf{WriteBucket}(\mathcal{P}(leaf_{ID}, \ell), \mathsf{S})$.

- **Evict (from Path ORAM).** This operation is to push blocks back from the stash S to the ORAM tree and keep the stash S occupancy low. First, it reads all the buckets along the lookup path, all the (remaining) real blocks are added to the stash S (i.e., ReadBucket). Meanwhile, the **Evict** operation writes as many blocks as possible from the stash S back to the lookup path, and the evicted blocks from the stash get pushed down as far as they can go (i.e., WriteBucket). This operation terminates after writing all real blocks from the stash S back to the lookup path, and each bucket is guaranteed to have Z real blocks. We denote the ReadBucket and WriteBucket functions as **Evict**$(leaf_{ID})$ operation, and the pseudocode is also shown in **EarlyReshuffles**$(leaf_{ID})$ (Line 2-4).

We briefly show the GetBlockOffset, ReadBucket, WriteBucket functions below, we refer reader to [22] for detailed descriptions.

- GetBlockOffset is to find the block of interest ID and return the permuted location of that block, or to return the permuted location of a random dummy block.
- ReadBucket is to read exactly Z real blocks in a bucket into the stash S. If the bucket contains less than Z real blocks, then the remaining blocks read out are updated dummy blocks (see **Main Invariants** below).
- WriteBucket is to evict up to Z blocks from stash S to a bucket. In particular, the location of $Z + S$ blocks are randomly reshuffled based on pseudo-random permutation or a truly random permutation. Eventually, the permuted data and its $offset$ are encrypted and written into the bucket.

Main Invariants. uORAM has three invariants, and we maintain these invariants at any time since they will determine the security of uORAM.

(1) Invariant 1. Every block is mapped to a leaf chosen uniformly at random in the ORAM tree $\mathcal{T}$. If a block ID is mapped to leaf $leaf_{ID}$, block ID is contained either in the stash S or in a bucket along the path from the root to leaf $leaf_{ID}$.
(2) Invariant 2. For every bucket in the tree $\mathcal{T}$, the physical position of the $Z+S$ real and dummy blocks in each bucket are randomly permuted with respect to all past and future writes to that bucket.
(3) Invariant 3. For every bucket along the path from the root to leaf $leaf_{ID}$, at least one dummy block in each bucket is randomly updated.

Data Access. uORAM takes $(op, ID, data^*, time)$ as input, outputs $data$.

(1) **EarlyReshuffles**$(leaf_{ID})$ $\triangleright$ from **Ring ORAM**
(2) $leaf_{ID} \leftarrow position[ID]$

(3) $position[ID] \leftarrow UniformRandom(0, 2^L - 1)$
(4) $data \leftarrow$ **ReadPath**$(leaf_{ID}, ID)$ $\triangleright$ from **Ring ORAM**
(5) $data \leftarrow$ read block ID from S
(6) if $op = read$ then return $data$ to client end if
(7) if $op = write$ then $\mathsf{S} \leftarrow (\mathsf{S} - \{(ID, data)\}) \bigcup \{(ID, data^*)\}$ end if
(8) **Evict**$(leaf_{ID}, time)$. $\triangleright$ from **Path ORAM**

To access a block ID, the client first invokes an operation **EarlyReshuffles**$(leaf_{ID})$ to read and write the real and dummy blocks on the path with leaf identifier $leaf_{ID}$ (Line 1). As a result, the client can determine the physical positions of a block of interest ID and dummy blocks per bucket. Second, the block of interest is remapped to a new random path and the position map is updated accordingly (Line 2 to 3). Third, the client invokes a read path operation **ReadPath**$(leaf_{ID}, ID)$ to read one block per bucket on that path, and then read that block into stash S (Line 4). If block ID is not found on path ℓ, it must be in Stash S (Line 5). Forth, if the access is read/write, then the client updates the content of block ID (Line 6 to 7). Last, the protocol invokes an eviction operation **Evict**$(leaf_{ID}, time)$ that reads all remaining real blocks on that path into stash S and writes the same path we just read from, percolating blocks down that path (Line 8). We remark that the dummy blocks in the read/write path can be updated using either randomness (w.r.t. **EarlyReshuffles** operation) or timestamp (w.r.t. **Evict** operation), both randomness and timestamp are chosen at random by client. We note that timestamp is linked to "time-locked" dummy block, which means we embed a time-lock in a dummy block using a timestamp *time* during **Evict** operation. In other words, the "time-locked" dummy block can be updated again once the specified timestamp *time* is reached (or "unlocked"). For example, another client may update the "unlocked" dummy block using a new randomness during his **EarlyReshuffles** operation.

3.1. Security Definition

We modify the standard ORAM security definition [23]. That is, adding a designated timestamp to the operation of an access pattern. Informally, two access patterns in a specific time-window (i.e., a time period that is considered best from starting to finishing some tasks such as user authentications) should be computationally indistinguishable. A crucial point is, the client makes data requests within an allowable time-window does not leak any useful information about the access pattern.

Definition 3.1 (Security Definition)**.** *Let*

$$\overleftarrow{y} = ((op_M, ID_M, data_M, time_M), \cdots, (op_1, ID_1, data_1, time_1))$$

denote a data sequence of M, where op_i denotes the i-th operation is a read or a write, ID_i denotes the address for that access, $data_i$ denotes the data being written, and $time_i$ denotes the designated timestamp such that $time_i \in \varrho$, where ϱ denotes an allowable time-window. Let uORAM($\overleftarrow{y}$) be the resulting sequence of operations between client and server under an uORAM protocol. The uORAM protocol guarantees that: 1) for any $\overleftarrow{y}$ and $\overleftarrow{y'}$, uORAM($\overleftarrow{y}$) and uORAM($\overleftarrow{y'}$) are computationally indistinguishable (by anyone except the client) if $|\overleftarrow{y}| = |\overleftarrow{y'}|$; 2) for any $\overleftarrow{y}$ the data returned to the client using uORAM is consistent with $\overleftarrow{y}$ (i.e., the uORAM behaves like a valid RAM) with overwhelming probability.

Security Analysis. We prove the security of uORAM using the similar approach described in [22].

Lemma 3.1. EarlyReshuffles *operation leaks no information.*

Proof. We let ReadBucket function read exactly $Z' = Z +$ "1" real blocks from random slots. If the bucket contains less than Z' real blocks, the remaining blocks read out are updated dummy blocks. The number "1" means that at least one dummy block in each bucket is updated by the client when reading. Meanwhile, the number of dummy blocks per bucket becomes $S' = S -$ "1". Similarly, WriteBucket function writes $Z' + S'$ (i.e., $Z + S$) encrypted blocks in a data-independent fashion. If there are $z' \leqslant Z'$ real blocks to be evicted to that bucket, then $Z' - z'$ updated dummy blocks are added. In particular, the $Z' + S'$ real and dummy blocks are randomly shuffled by the client. Overall, $\mathcal{A}$ learns nothing during **EarlyReshuffles** operation. □

Lemma 3.2. ReadPath *operation leaks no information.*

Proof. The path selected for reading will look random to $\mathcal{A}$ due to Invariant 1 such that leaves are chosen uniformly at random. From Invariant 2, we know that every bucket is randomly reshuffled. In particular, the real and updated dummy blocks we read are indistinguishable to $\mathcal{A}$ due to Invariant 3 (the Lemma 2.3 confirms such fact). Therefore, $\mathcal{A}$ learns nothing during **ReadPath** operation. □

Lemma 3.3. Evict *operation leaks no information.*

Proof. The path selected for eviction is chosen uniformly and is public. ReadBucket function reads all remaining real and updated dummy blocks from the bucket stored on the server. The real and updated dummy blocks on the lookup path are stored into stash S after ReadBucket. WriteBucket function writes the real and updated dummy blocks from the stash S into the specified bucket on the server. In particular, the client writes back all updated dummy blocks to their original format. Meanwhile, the client erases her random permutation in the **Bucket Metadata**. If there is $z \leqslant Z$ real blocks to be evicted to that bucket, then $Z - z$ "time-locked" blocks are added. The "time-locked" blocks are the blocks that cannot be updated until a designated timestamp is reached. We remark that the real and "time-locked" blocks we write back are indistinguishable to $\mathcal{A}$ because the specified timestamp *time* is randomly chosen by the client (otherwise, the server may keep a history of the updated dummy blocks if a client updates them multiple times within a time-window ϱ, thus break the unlinkability of uORAM). Therefore, $\mathcal{A}$ learns nothing during **Evict** operation within ϱ. □

3.2. Stash and Bandwidth Analysis

Since a small number of blocks might overflow from the tree bucket on the server after **Evict** operation, the client locally stores these overflowing blocks in the stash S. To analyze the stash usage (i.e., measuring the number of real blocks that remain in the stash after **Evict** operation), we rely on the theoretical result of Path ORAM [21]. The Theorem 1 in [21] has shown that the probability (i.e., the number of real blocks in the stash exceeds R) decreased exponentially in R (i.e., the stash size).

Now, we analyze the overall bandwidth of proposed uORAM, including **EarlyReshuffles**, **ReadPath** and **Evict** operations. The **EarlyReshuffles** operation reads exactly $Z +$ "1" real blocks and writes back $Z + S$ real and dummy blocks per bucket, so the bandwidth is $(2Z + S +$ "1"$)(L + 1)$ ($L = \log N$). The number "1" means that at least one dummy block per bucket is updated by the client due to Invariant 3.

We observe that the **ReadPath** operation first transfers $L+1$ blocks — one from each bucket. Then, the client reads the remaining real and updated dummy blocks into the stash and writes back $Z+S$ real and dummy blocks per bucket during **Evict** operation. So, the overall bandwidth is $2(2Z+S+\text{“1”})(L+1)$. We notice that the overall bandwidth of uORAM is higher than a Path or Ring ORAM. However, neither Path ORAM nor Ring ORAM can achieve our design goal (see Appendix A).

4. The Proposed General Framework

In this section, we first present the security models (user authenticity and strong privacy) for our proposed pBRUA. Then, we show the proposed general framework of pBRUA.

States. We define a system user set $\mathcal{U}$ with n users, i.e. $|\mathcal{U}| = n$. We say an instance oracle $\Pi^i_{\mathtt{pk}}$ (i.e., session i of user $\mathtt{pk}$) may be used or unused, and a user $\mathtt{pk}$ has an unlimited number of instances called oracles. The oracle is considered as unused if it has never been initialized. Each unused oracle $\Pi^i_{\mathtt{pk}}$ can be initialized with a secret key $\mathtt{sk}$. The oracle is initialized as soon as it becomes part of a group. After the initialization the oracle is marked as used and turns into the stand-by state where it waits for an invocation to execute a protocol operation. Upon receiving such invocation, the oracle $\Pi^i_{\mathtt{pk}}$ learns its partner id $\mathsf{pid}^i_{\mathtt{pk}}$ (i.e., a collection of recognized users by the instance oracle $\Pi^i_{\mathtt{pk}}$) and turns into a processing state where it sends, receives and processes messages according to the description of the protocol. During that stage, the internal state information $state^i_{\mathtt{pk}}$ is maintained by the oracle. The oracle $\Pi^i_{\mathtt{pk}}$ remains in the processing state until it collects enough information to finalize the user authentications. As soon as the user authentication is accomplished, $\Pi^i_{\mathtt{pk}}$ accepts and terminates the protocol execution meaning that it would not send or receive further messages. If the protocol execution fails then $\Pi^i_{\mathtt{pk}}$ terminates without having accepted. We define $\mathsf{sid}^i_{\mathtt{pk}}$ as the unique session identifier belonging to user $\mathtt{pk}$ of session i. Specifically, $\mathsf{sid}^i_{\mathtt{pk}} = \{m_j\}_{j=1}^n$, where $m_j \in \{0,1\}^*$ is the message transcript among users in $\mathsf{pid}^i_{\mathtt{pk}}$. The session identifier means that the session which $\Pi^i_{\mathtt{pk}}$ participates in is defined by a unique session id $\mathsf{sid}^i_{\mathtt{pk}}$, and this value is known to all oracles participating in the same session.

4.1. Definition

A pBRUA framework consists of the following algorithms:

- Setup: The algorithm takes a security parameter λ as input, outputs a public parameter pp.
- Enrollment: This is a non-interactive protocol between a user and an authentication server in a secure channel. The user first generates a secret/public key pair $(\mathtt{sk},\mathtt{pk})$ using public parameter pp, derives a sketch $\mathsf{SS}(s,w)$ from her biometrics w and a secret string s. Then, she enrolls her identity ID, public key $\mathtt{pk}$ and sketch $\mathsf{SS}(s,w)$ to the authentication server. The enrolled users become authorized ones after enrollment. We assume a uniform[3] biometrics source $\mathcal{M}$ and $w \in \mathcal{M}$.
- Authentication: This is an interactive protocol between an authorized user and an authentication server over a public channel. An authorized user takes public parameter pp, her nearby biometrics w' and her enrolled sketch $\mathsf{SS}(s,w)$ as input, outputs a message-signature pair (m,σ). The authentication server accepts the user if the message-signature pair (m,σ) is verified as valid under her enrolled public key

[3]One may question a uniform source is not practical, we stress that the uniform source can be replaced by a non-uniform source (e.g., symbol-fixing source [39]) while the security of FE is held. We use a uniform source here just for simplicity, and the case of the non-uniform source was explicitly discussed in [10, 12].

$\mathtt{pk}$. The message m contains the ephemeral data transmitted during user authentication, and the nearby biometrics satisfies $\mathsf{dist}(w', w) \leqslant t$.

4.2. Security Models

A secure pBRUA framework should achieve several security goals: user authenticity and user privacy. Below we present corresponding security models to capture these requirements.

Authenticity. Informally, an adversary attempts to impersonate an authorized user and authenticate herself to an authentication server. We define a formal authenticity game between a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ below.

- Setup. $\mathcal{C}$ first generates public/secret key pairs $\{(\mathtt{pk}_i \mathtt{sk}_i)\}_{i=1}^n$ ($i \in \{1, n\}$) for n users and an authentication server $\mathbb{S}$ in the system. $\mathcal{C}$ also generates biometrics information w_i and its corresponding sketch $\mathsf{SS}(s_i, w_i)$ for individual users. Eventually, $\mathcal{C}$ sends all users' public keys and sketches to $\mathcal{A}$.
- Training. $\mathcal{A}$ can make the following queries in an arbitrary sequence to $\mathcal{C}$.
  - ∗ Send: If $\mathcal{A}$ issues a send query in the form of $(\mathtt{pk}, i, m)$ to simulate a network message for the i-th session of user $\mathtt{pk}$, then $\mathcal{C}$ would simulate the reaction of instance oracle $\Pi^i_{\mathtt{pk}}$ upon receiving message m, and return to $\mathcal{A}$ the response that $\Pi^i_{\mathtt{pk}}$ would generate; If $\mathcal{A}$ issues a send query in the form of $(\mathtt{pk}', 'start')$, then $\mathcal{C}$ creates a new instance oracle $\Pi^i_{\mathtt{pk}'}$ and returns to $\mathcal{A}$ the first protocol message.
  - ∗ Biometrics Reveal: If $\mathcal{A}$ issues a biometrics reveal query to user i, then $\mathcal{C}$ returns user i's biometrics information w_i to $\mathcal{A}$.
  - ∗ Secret Key Reveal: If $\mathcal{A}$ issues a secret key reveal query to user i, then $\mathcal{C}$ returns user i's enrolled secret key $\mathtt{sk}_i$ to $\mathcal{A}$.
- Challenge. $\mathcal{A}$ wins the game if all of the following conditions hold.

  (1) $\mathbb{S}$ accepts user $\mathtt{pk}_i$. It implies $\mathsf{sid}^s_{\mathbb{S}}$ (i.e., session s of server $\mathbb{S}$) exist.
  (2) $\mathcal{A}$ issued neither Biometrics Reveal nor Secret Key Reveal query to user $\mathtt{pk}_i$.
  (3) $m \in \mathsf{sid}^s_{\mathbb{S}}$, but there exists no instance oracle $\Pi^s_{\mathtt{pk}_i}$ which has sent m (m denotes the message transcript from user $\mathtt{pk}_i$).

  We define the advantage of an adversary $\mathcal{A}$ in the above game as

$$\mathtt{Adv}_{\mathcal{A}}(\lambda) = |\Pr[\mathcal{A} \textit{ wins}]|.$$

Definition 4.1. *We say a pBRUA protocol has user authenticity if for any PPT $\mathcal{A}$, $\mathtt{Adv}_{\mathcal{A}}(\lambda)$ is a negligible function of the security parameter λ.*

Strong Privacy. Informally, an authentication server $\mathbb{S}$ is not allowed to identify who are the authorized users in a certain time-window. We define a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ as follows:

- Setup: $\mathcal{C}$ generates public/secret key pairs $\{(\mathtt{pk}_i \mathtt{sk}_i)\}_{i=1}^n$ for n users and an authentication server $\mathbb{S}$ in the system. In addition, $\mathcal{C}$ generates biometrics information w_i and its corresponding sketch $\mathsf{SS}(s_i, w_i)$ for individual users. Eventually, $\mathcal{C}$ sends all public information to $\mathcal{A}$, and the authentication server $\mathbb{S}$ is completely controlled by $\mathcal{A}$. $\mathcal{C}$ tosses a random coin b which will be used later in the game. We denote the original n users set as $\mathcal{U}$.

- Training: $\mathcal{A}$ is allowed to issue Send query and at most n-2 Secret Key Reveal and Biometrics Reveal queries to $\mathcal{C}$. We denote $\mathcal{U}'$ as the set of honest users whose biometrics as well as secrets keys are not corrupted.
- Challenge: $\mathcal{A}$ randomly selects two users $\mathtt{pk}_i, \mathtt{pk}_j \in \mathcal{U}'$ as challenge candidates, then $\mathcal{C}$ removes them from $\mathcal{U}'$ and simulates $\mathtt{pk}_b^*$ to $\mathcal{A}$ by either $\mathtt{pk}_b^* = \mathtt{pk}_i$ if $b = 1$ or $\mathtt{pk}_b^* = \mathtt{pk}_j$ if $b = 0$. $\mathcal{C}$ chooses a time-window ϱ, lets $\mathcal{A}$ interact with challenge user $\mathtt{pk}_b^*$ in the time-window ϱ.

$$\mathcal{A} \Leftrightarrow \mathtt{pk}_b^* = \begin{cases} \mathtt{pk}_i & b = 1 \\ \mathtt{pk}_j & b = 0 \end{cases}$$

Finally, $\mathcal{A}$ outputs b' as its guess for b. If $b' = b$, then $\mathcal{C}$ outputs 1; Otherwise, $\mathcal{C}$ outputs 0. We define the advantage of $\mathcal{A}$ in the above game as

$$\mathtt{Adv}_{\mathcal{A}}(\lambda) = \Pr[\mathcal{C} \to 1] - 1/2.$$

Definition 4.2. *We say a pBRUA protocol has strong privacy if for any PPT $\mathcal{A}$, $\mathtt{Adv}_{\mathcal{A}}(\lambda)$ is a negligible function of the security parameter λ.*

4.3. Proposed Construction

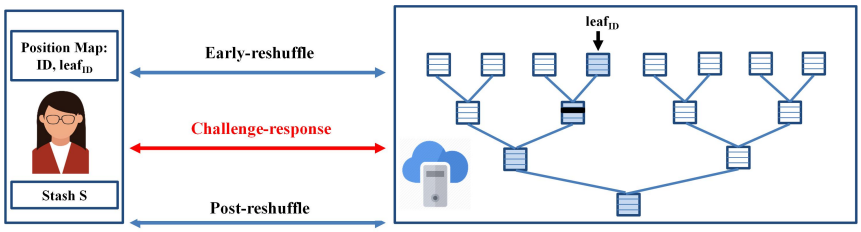

Fig. 3. High-level user authentication via uORAM.

High-level Description. Suppose at most n users enroll themselves to an authentication server, and the authentication server built a binary tree to store all users' enrolled information. Next, an uORAM protocol is executed between an authorized user and the authentication server for user authentications (including early-reshuffle phase, challenge-response phase, and post-reshuffle phase), which is described in Figure 3. Specifically, in the early-reshuffle phase, an authorized user randomizes and permutes either her real block or a dummy block in each bucket along a tree path (i.e., **EarlyReshuffles** operation, and the tree path must include her real block). In the challenge-response phase, the authorized user first obtains the permutation of either her real block or a dummy block in each bucket by performing the **ReadPath** operation. Then, the authorized user derives a message-signature pair based on her real block and the randomized dummy blocks for proving her authenticity. In the post-reshuffle phase, the authorized user re-randomizes all blocks in the tree path (i.e., **Evict** operation). The proposed pBRUA framework is mainly used to achieve strong privacy for valid user authentications. The proposed pBRUA framework consists of the following building blocks. Meanwhile, we note that a blind signature scheme [40] could be used to prevent an unauthorized user with NO enrolled biometrics from performing a valid user authentication in the pBRUA framework (the detailed explanation is referred to Remark 2).

- A LWE-based computational fuzzy extractor $\mathsf{FE} = (\mathsf{Gen}, \mathsf{Rep})$.
- An Existential Unforgeability under Chosen Message Attack EUF-CMA secure digital signature $\Sigma = (\mathsf{Setup}, \mathsf{KG}, \mathsf{Sign}, \mathsf{Verify})$.
- An Indistinguishability of Keys under Chosen Plaintext Attack IK-CPA secure public key encryption $\mathsf{PKE} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$.
- The uORAM protocol.

Block Structures. The real block consists of two parts: user i's public key and sketch $(\mathtt{pk}_2, \mathsf{SS}(s_2, w_2))$. The dummy block is of the form $(\mathtt{pk}_1, \mathsf{SS}(s_1, \mathit{empty}))$, where $\mathtt{pk}_1$ denotes a random public key, the data in sketch SS is *empty* (i.e., "0") and the secret string s_1 is chosen at random. We stress that the dummy block is a public resource during $\mathsf{Enrollment}$, it becomes a private one during $\mathsf{Authentication}$. The status of real block $(\mathtt{pk}_2, \mathsf{SS}(s_2, w_2))$ and dummy block $(\mathtt{pk}_1, \mathsf{SS}(s_1, 0))$ at different phases are shown in Figure 4, respectively.

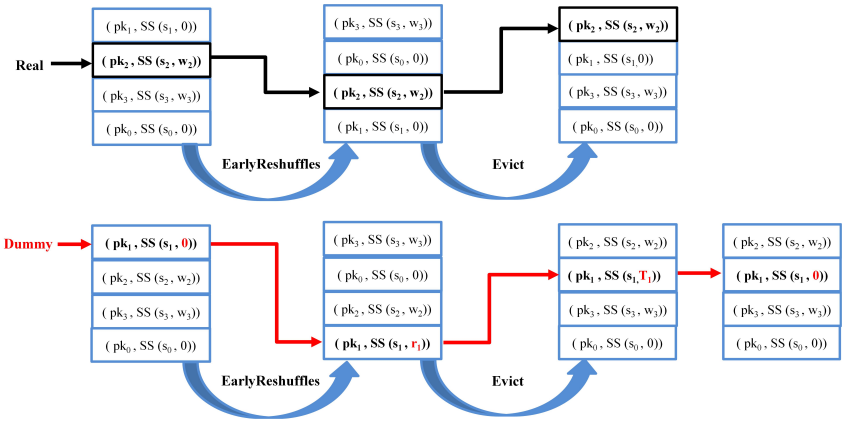

Fig. 4. The status of real or dummy block in a bucket. The physical position of real or dummy block in the bucket will be permuted at random under **EarlyReshuffles** operation and **Evict** operation, respectively. Meanwhile, the hidden message (r_1 denotes the randomness and T_1 denotes the timestamp) in dummy sketch will be updated accordingly.

Our Construction. Let $\{\mathtt{H}_z : \mathbb{Z}_q^{n_0} \to \mathbb{Z}_q\}$ be a universal hash function with linearity that we reviewed in Section 2.3. For each seed $z \in \mathbb{Z}_{q^{n_0}}$ and $y \in \mathbb{Z}_q$, we define "$\mathtt{H}_z^{-1}(y)$" as the set of pre-images of y under $\mathtt{H}_z$. That is, $\mathtt{H}_z^{-1}(y) = \{x \in \mathbb{Z}_q^{n_0} : \mathtt{H}_z(x) = y\}$. In particular, $x \xleftarrow{\mathrm{R}} \mathtt{H}_z^{-1}(y)$ means that we choose an element x uniformly from the set $\mathtt{H}_z^{-1}(y)$, and its dimension is n_0. We run the Setup algorithm first to generate the public parameter pp in the system. Then, we use a user i and an authentication server $\mathbb{S}$ to illustrate our general framework.

- $\mathsf{Enrollment}$. A user i enroll herself to an authentication server $\mathbb{S}$ will perform the following
  - generate a secret/public key pair $(\mathtt{sk}_i, \mathtt{pk}_i) \leftarrow \Sigma.\mathsf{KG}(pp)$.
  - obtain a secret/public string pair $(s_i, p_i) \leftarrow \mathsf{FE.Gen}(pp, w_i)$, where $s_i \xleftarrow{\mathrm{R}} \mathtt{H}_z^{-1}(\mathtt{sk}_i)$, $w_i \leftarrow \mathcal{M}$ (the biometrics distribution $\mathcal{M}$ is referred to Section 2.4), and public string $p_i = \mathsf{SS}(s_i, w_i)$.
  - send public values $(\mathtt{pk}_i, \mathsf{SS}(s_i, w_i))$ to $\mathbb{S}$.

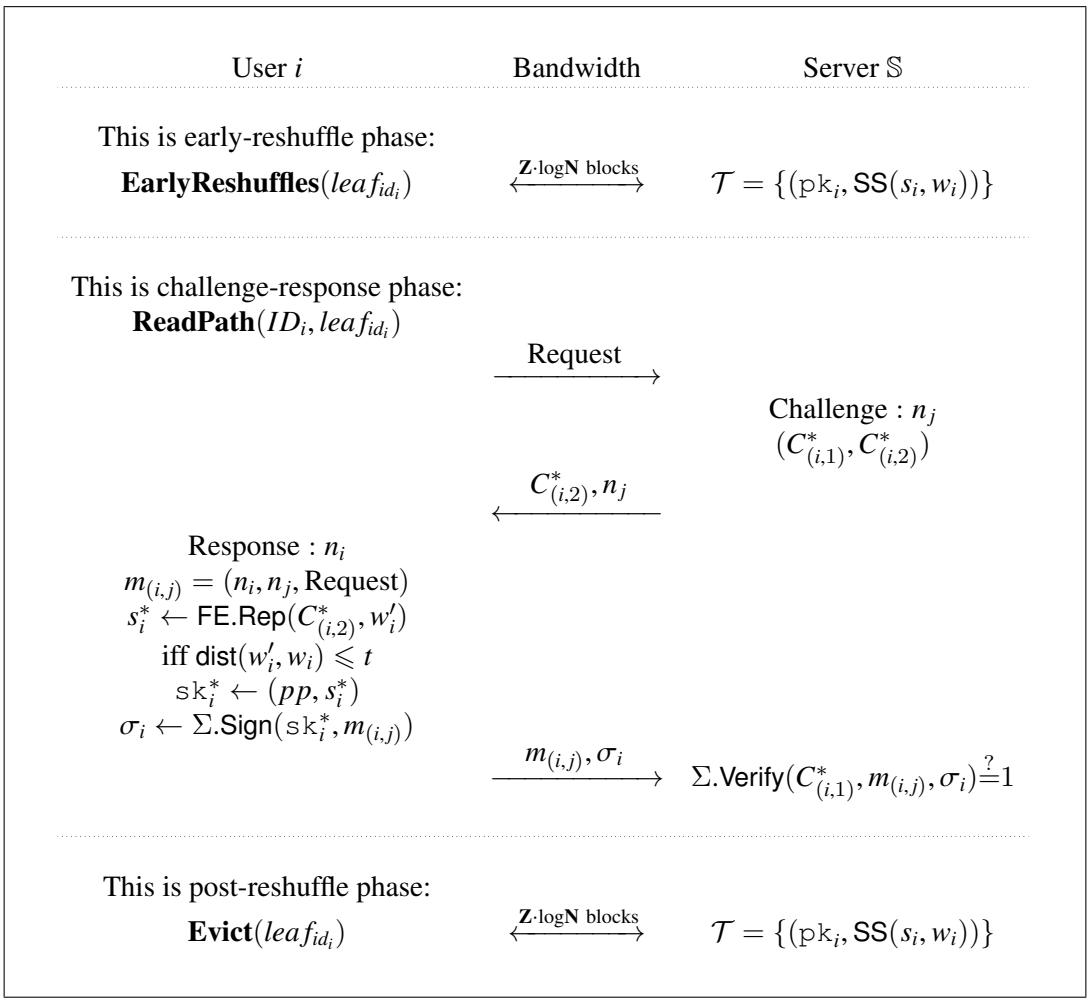

Fig. 5. The Authentication includes early-reshuffle, challenge-response, and post-reshuffle phases. The bandwidth of early-reshuffle phase and post-reshuffle phase are $Z \cdot \log N$, respectively. In the challenge-response phase, server $\mathbb{S}$ returns a ciphertext $C^*_{i,2}$ (i.e., aggregated "sketch") only.

According to the uORAM protocol, $\mathbb{S}$ regards user i's public information $(\mathtt{pk}_i, \mathsf{SS}(s_i, w_i))$ as an enrolled record, stores into a bucket and returns the leaf identifier $leaf_{id_i}$ of that tree path to user i. The user i can identify her block of interest in a bucket of the lookup path using leaf identifier $leaf_{id_i}$ and block identifier ID_i, where $leaf_{id_i} \leftarrow position[ID_i]$ and ID_i is the secret block identifier.

- Authentication. The detailed interaction between a user i and authentication server $\mathbb{S}$ is described in Figure 5. We use the previously mentioned three phases to present the general framework.

  $*$ Early-reshuffle Phase: User i performs the **EarlyReshuffles**$(leaf_{id_i})$ operation to update the $offset_j$ $(j \in \{0, L\})$ for all buckets along the lookup path. Specifically, user i: 1) updates a dummy block as $(\mathtt{pk}_j, \mathsf{SS}(s_j, r_j))$ in a bucket, where the randomness $r_j \leftarrow \mathcal{M}$ is chosen at random by user i (see remark below for detailed description); 2) writes $offset$ into $\mathbf{BucketMetadata}_j \leftarrow \mathsf{PKE.Enc}(\mathtt{pk}_i, offset_j)$ per bucket along the lookup path.

* Challenge-response Phase: User i performs the **ReadPath**$(leaf_{id_i}, ID_i)$ operation to obtain the random $offset_j \leftarrow \mathsf{PKE.Dec}(\mathtt{sk}_i, \textbf{BucketMetadat}_j)$ per bucket in the lookup path. Upon receiving an authentication request (i.e., a set of offset $\{offset_j\}_{j=0}^{L}$ and leaf identifier $leaf_{id_i}$) from user i, $\mathbb{S}$ computes an "aggregated" version of requested blocks, which consists of a (L+1 size) set of public keys and sketches: $C^*_{(i,1)} \leftarrow \prod_{i=0}^{L} \mathtt{pk}_i, C^*_{(i,2)} \leftarrow \sum_{i=0}^{L}(\mathsf{SS}(s_i, w_i))$ (see correctness below), and returns a single ciphertext $C^*_{(i,2)}$ and a challenge randomness n_j to user i.
* Challenge-response Phase: User i performs the following
  (1) choose a response randomness n_i and generate the message $m_{(i,j)} = (n_i, n_j, \text{Request})$.
  (2) extract the secret string by running algorithm $s^*_i \leftarrow \mathsf{FE.Rep}(C^*_{(i,2)}, w'_i)$ iff $\mathsf{dist}(w'_i, w_i) \leqslant t$, and obtain the "aggregated" secret key $\mathtt{sk}^*_i \leftarrow \mathtt{H}_z(s^*_i)$.
  (3) generate a message-signature pair $(m_{(i,j)}, \sigma_i) \leftarrow \Sigma.\mathsf{Sign}(\mathtt{sk}^*_i, m_{(i,j)})$ and send it to $\mathbb{S}$.
* Challenge-response Phase: $\mathbb{S}$ verifies $1 \stackrel{?}{=} \Sigma.\mathsf{Verify}(C^*_{(i,1)}, m_{(i,j)}, \sigma_i)$ under the "aggregated" public key $C^*_{(i,1)}$. If the signature passes the verification, it accepts; Otherwise, it aborts.
* Post-reshuffle Phase: User i performs the **Evict**$(leaf_{id_i})$ operation. Specifically, **Evict**$(leaf_{id_i})$ operation reads all remaining real blocks along the lookup path into the local stash S, writes real and updated dummy blocks back to the lookup path.

Instantiations. We hereby try to instantiate the underlying cryptographic building blocks. First, to instantiate the LWE-based computational fuzzy extractors, we could use the fuzzy extractor scheme in [10–12]. In particular, we require that all enrolled users share the common public parameters, just like the first reusable fuzzy extractor scheme in [11]. Second, we use the Schnorr signature scheme with homomorphic property to instantiate the underlying digital signatures. Meanwhile, the Waters signature scheme described in [19] should be also suitable for our general framework. Last, the public key encryption scheme (which was used in **Bucket Metadata**) can be instantiated to ElGamal cryptosystem [41] for achieving IK-CPA security, and we refer readers to [42] for Cramer-Shoup cryptosystem [43] with IK-CCA security which might be alternatively applicable to instantiate our general framework.

Correctness. In the challenge-response phase of user authentication, the returned block is either a block of interest or an updated dummy block in a bucket. Specifically, **ReadPath** opreation will request L+1 blocks in a tree path—one from each bucket. That is, $(\mathtt{pk}_0, \mathsf{SS}(s_0, r_0)), (\mathtt{pk}_i, \mathsf{SS}(s_i, w_i)), \cdots, (\mathtt{pk}_L, \mathsf{SS}(s_L, r_L))$, where $\mathtt{pk}_i = g^{\mathtt{sk}_i}$ and $\mathsf{SS}(s_i, w_i) = \mathbb{A} \cdot s_i + w_i$ ($g, \mathbb{A} \in pp$). We notice that a dummy block after **EarlyReshuffles** operation will be updated as $(\mathtt{pk}_0, \mathsf{SS}(s_0, r_0)) = (g^{\mathtt{sk}_0}, \mathbb{A} \cdot s_0 + r_0)$, where randomness r_0 is independently chosen and stored by user i (see remark below). To achieve a constant bandwidth, the authentication server $\mathbb{S}$ returns a single "aggregated" ciphertext (underline part) to user i: $(C^*_{(i,1)}, \underline{C^*_{(i,2)}})$, where $C^*_{(i,1)} = \mathtt{pk}_0 \cdot \mathtt{pk}_1 \cdots \mathtt{pk}_L$, $\underline{C^*_{(i,2)}} = \mathsf{SS}(s_0, r_0) + \mathsf{SS}(s_i, w_i) + \cdots + \mathsf{SS}(s_L, w_L)$ and $\cdot$ denotes the multiplication. More specifically,

$$C^*_{(i,1)} = \mathtt{pk}_0 \cdot \mathtt{pk}_1 \cdots \mathtt{pk}_L = g^{\sum_{i=0}^{L}(\mathtt{sk}_i)}$$

$$\underline{C^*_{(i,2)}} = \mathsf{SS}(s_0, r_0) + \mathsf{SS}(s_i, w_i) + \cdots + \mathsf{SS}(s_L, r_L) = \mathbb{A}(\sum_{i=0}^{L}(s_i)) + \sum_{i=0}^{L-1}(r_i) + w_i$$

where $\sum_{i=0}^{L}(\mathtt{sk}_i) = \mathtt{H}_z(\sum_{i=0}^{L}(s_i))$. The user i can obtain the "aggregated" sketch (includes her interested sketch $\mathsf{SS}(s_i, w_i)$) by removing the randomness $\{r_i\}_{i=0}^{L-1}$, and the challenge-response of user authentications proceeds as the protocol specified.

Remark 1. The **ReadPath** operation relies on a metadata: **Bucket Metadata**. We use an example to illustrate its workflow, and we assume a bucket includes 4 blocks. We also assume the bucket includes a block interested by user i $(\mathtt{pk}_i, \mathsf{SS}(s_i, w_i))$, a block interested by another user j $(\mathtt{pk}_j, \mathsf{SS}(s_j, w_j))$ and two dummy blocks $(\mathtt{pk}_0, \mathsf{SS}(s_0, 0)), (\mathtt{pk}_1, \mathsf{SS}(s_1, 0))$, where s_0, s_1 are chosen at random by server $\mathbb{S}$.

The **EarlyReshuffles** operation includes ReadBucket and WriteBucket functions. The ReadBucket function will read all real blocks from the bucket into the local stash. Specifically, a user i finds her block of interest (**interest**) and one dummy block (**dummy**) per bucket in a "compute and compare" manner. In particular, user i updates one dummy block (**updated**). More concretely,

$$\mathtt{pk}_i = g^{\mathtt{sk}_i}, \mathtt{sk}_i \leftarrow \mathsf{FE.Rep}(\mathsf{SS}(s_i, w_i), w_i'). \triangleright \textbf{interest}$$

$$\mathtt{pk}_0 = g^{\mathtt{sk}_0}, \mathtt{sk}_0 \leftarrow \mathsf{FE.Rep}(\mathsf{SS}(s_0, 0), 0). \triangleright \textbf{dummy}$$

$$\mathtt{pk}_0' = g^{\mathtt{sk}_0'}, \mathtt{sk}_0' \leftarrow \mathsf{FE.Rep}(\mathsf{SS}(s_0, r_0), r_0). \triangleright \textbf{updated}$$

where randomness $r_0 \leftarrow \mathcal{M}$ is chosen by user i and the updated dummy block can be decrypted by user i only. Afterwards, use i writes back 3 blocks (i.e., one updated dummy block $\mathtt{pk}_0'$ and two real blocks $\mathtt{pk}_i$ and $\mathtt{pk}_j$) and writes the permuted $offset$ into the **Bucket Metadata** (i.e., WriteBucket function). Specifically, user i chooses the $offset \in \{0, \cdots, 3\}$ at random and encrypts the random $offset$ by running $C_i \leftarrow \mathsf{PKE.Enc}(\mathtt{pk}_i, offset)$. Later, if user i performs the **ReadPath** operation, then she can obtain the random $offset \leftarrow \mathsf{PKE.Dec}(\mathtt{sk}_i, C_i)$.

The **Evict** operation requires that the block of interest ID_i must be re-randomized as $(\mathtt{pk}_i', \mathsf{SS}(s_i', data)) = (g^{\mathtt{sk}_i'}, \mathbb{A} \cdot s_i' + data)$, where $\mathtt{sk}_i'$ is chosen at random and $s_i' = \mathtt{H}_z^{-1}(\mathtt{sk}_i')$. Other blocks (include dummy blocks) in the same bucket should also be re-randomized accordingly. For example, a block $(\mathtt{pk}_j, \mathsf{SS}(s_j, w_j))$ is re-randomized as $(\mathtt{pk}_j', \mathsf{SS}(s_j', w_j) = (\mathtt{pk}_j \cdot g^{\Delta(\mathtt{sk}_j)}, \mathsf{SS}(s_j, w_j) + \mathbb{A} \cdot \Delta(s_j))$, where $\Delta(s_j) \leftarrow \mathtt{H}_z^{-1}(\Delta(\mathtt{sk}_j))$, and $\Delta(\mathtt{sk}_j)$ is chosen at random. Note that the $data$ can be a new biometrics such as $\overline{w_i} \neq w_i$. According to the design of uORAM protocol, the dummy block is a common resource shared by all enrolled users, user i should remove her randomness r_0 in sketch $\mathsf{SS}(s_0, r_0)$. To maintain the security of Definition 3.1, user i updates the dummy block to a "time-locked" format: $\mathsf{SS}(s_0, time_0) = \mathbb{A} \cdot s_0 + time_0$, where $time_0 \leftarrow \mathcal{M}$ denotes a designated timestamp. Once the designated timestamp is reached, the "time-locked" dummy block becomes a public one. We note that an extra (mapping) mechanism could be used to transform a timestamp with the standard format into a distribution over $\mathbb{Z}_q^{m_0}$.

Since multiple users share an uORAM protocol and each user has an individual stash, the user should push all real blocks back to the uORAM tree during **Evict** operation. This is because if real blocks reside in the local stash S, then user authentication may fail. We leave it as a future work to ensure a policy such that the user must push real blocks back to the uORAM tree, or to ensure a valid user authentication even when real blocks are resided in the stash S.

Remark 2. One may notice that an unauthorized (or unenrolled) user with NO biometrics data can be successfully authenticated by the authentication server. This is because the dummy blocks in the tree are common resources, any third parties can distinguish them from real blocks (as described previously). In this way, the unauthorized user updates at least one dummy block in each bucket in a tree path, and utilizes them for generating a valid message-signature pair and authenticating herself to the authentication server. We stress that such security threat is independent from the security concern in our proposed user authenticity model (an adversary tries to impersonate an authorized user). In particular, the unauthorized user authentication will NOT help the adversary (e.g., impersonator) to win the user authenticity game.

To tackle such security threat, we rely on an extra cryptographic tool: signatures on randomizable ciphertexts [40], such that given a signature on a ciphertext, any third parties (know neither the signing key nor the encrypted message) can randomize the ciphertext and adapt the signature to the re-randomized ciphertext. A pair of ciphertext and its signature can be randomized simultaneously and consistently. In particular, a given signature can be transformed into a new one on the same message, which in turn yields a blind signature scheme.

The modified pBRUA framework is described as follows: an authorized user runs the interactive blind signature scheme proposed in [40], to derive a publicly verifiable ciphertext-signature pair (C, σ) during Enrollment. Let the encrypted message be denoted as ID_i (i.e., user i's secret block identifier). In the challenge-response phase of Authentication, an authorized user sends the derived ciphertext-signature pair to the authentication server as the authentication request. The authentication executes the pBRUA protocol if the ciphertext-signature pair is valid (i.e., the anonymous authorized user is an enrolled one). We stress that, the authentication server still cannot link the anonymous authorized user across different authentication sessions, because the ciphertext-signature pair has the blindness, and it can be randomzied by the authorized user with the same encrypted message ID_i.

5. Security Analysis

In this section, we show the security result of our proposed pBRUA framework.

Theorem 5.1. *The proposed pBRUA achieves user authenticity if the D-LWE$_{q,n_0-k,m_0,\chi}$ assumption is (ϵ, s'_{sec}) secure, the family of universal hash functions $\mathbb{H} \leftarrow \{\mathtt{H}_z : \mathbb{Z}_q^{n_0} \to \mathbb{Z}_q\}_{z\in Z}$ is ϵ-statistically secure and the digital signature scheme Σ is EUF-CMA secure.*

Proof. We define a sequence of games $\{\mathbb{G}_i\}$ and let $\mathtt{Adv}_i^{\text{pBRUA}}$ denote the advantage of the adversary $\mathcal{A}$ in game $\mathbb{G}_i$. Assume that $\mathcal{A}$ activates at most m sessions in each game. We highlight the differences between adjacent games by underline. For simplicity, we ignore the technique for constant bandwidth in the following and subsequent proofs.

- $\mathbb{G}_0$: This is the original game for user authenticity security.
- $\mathbb{G}_1$: This game is identical to game $\mathbb{G}_0$ except that the challenger $\mathcal{C}$ will output a random bit if the authentication server $\mathbb{S}$ accepts a user i, but $\mathsf{sid}_i^s \neq \mathsf{sid}_{\mathbb{S}}^s$ (i.e., a session s between user i and server $\mathbb{S}$). Since n users involved in this game, we have:

$$\left|\mathtt{Adv}_0^{\text{pBRUA}} - \mathtt{Adv}_1^{\text{pBRUA}}\right| \leqslant n \cdot m^2/2^{\lambda}. \tag{1}$$

- $\mathbb{G}_2$: This game is identical to game $\mathbb{G}_1$ except the following difference: $\mathcal{C}$ randomly chooses $g \in \{1, m\}$ as a guess for the index of the Challenge session. $\mathcal{C}$ will output a random bit if $\mathcal{A}$'s challenge query does not occur in the g-th session. Therefore we have

$$\mathtt{Adv}_1^{\text{pBRUA}} = m \cdot \mathtt{Adv}_2^{\text{pBRUA}}. \tag{2}$$

- $\mathbb{G}_3$: This game is identical to game $\mathbb{G}_2$ except that in the g-th session, the k-size pseudorandom bit of hidden secret in the sketch $\mathsf{SS}(s_i, w_i)$ of user i is replaced by a random value U. Below we show that the difference between $\mathbb{G}_2$ and $\mathbb{G}_3$ is negligible under the D-LWE$_{q,n_0-k,m_0,\chi}$ assumption.

Let $\mathcal{C}$ denote a distinguisher against the D-LWE$_{q,n_0-k,m_0,\chi}$ assumption, who is given a tuple $(\underline{X_{1,\cdots,k}}, \mathbb{A}, \mathbb{A}\cdot X+\chi)$, aims to distinguish the real LWE tuple from a random tuple $(\underline{U}, \mathbb{A}, \mathbb{A}\cdot X+\chi)$ where $U \in \mathbb{Z}_q^k$. $\mathcal{C}$ simulates the game for $\mathcal{A}$ as follows.

* Setup. $\mathcal{C}$ sets up the game for $\mathcal{A}$ by creating n users with the corresponding block identifiers $\{ID_i\}$. $\mathcal{C}$ randomly selects index i and guesses that the g-th session will happen with regard to user i. $\mathcal{C}$ sets the sketch of user i as $\mathsf{SS}(s_b, w_i)$ such that $\mathsf{SS}(s_b, w_i) = \mathbb{A}\cdot\underline{X_b}+\chi$, where $X_b = \underline{X_{1,\cdots,k}}$. $\mathcal{C}$ sets user i's enrolled secret key as $\mathtt{sk}_b \leftarrow \mathtt{H}_z(\underline{X_b})$ (its public key is $\mathtt{pk}_b \leftarrow \Sigma.\mathsf{KG}(pp, \mathtt{sk}_b)$). $\mathcal{C}$ honestly generates biometrics, public/secret key pairs and sketches as Enrollment specified for n-1 users. In addition, $\mathcal{C}$ generates certain dummy public keys and sketches in the system. Eventually, $\mathcal{C}$ sends all real/dummy enrolled public keys and references to $\mathcal{A}$. Note that we choose a random vector from $\mathbb{Z}_q^{n_0-k}$ to generate $X_b \in \mathbb{Z}_q^{n_0}$, we omit it in the following proof.
* Training. $\mathcal{C}$ answers $\mathcal{A}$'s queries as follows.
  * If $\mathcal{A}$ issues a Send query in the form of $(n_j, C^*_{(i,2)})$ to $\mathcal{C}$, where $C^*_{(i,2)}$ includes a re-randomized real sketch $\mathsf{SS}(s'_b, w_i)$ and L dummy sketches $\{\mathsf{SS}(s_j, r_j)\}$. $\mathcal{C}$ chooses a response randomness n_i first, then $\mathcal{C}$ honestly generates the protocol transcript T_i using user i's enrolled secret key $\mathtt{sk}_b$. Specifically, $T_i = (m_{(i,j)}, \sigma_i)$, where $\sigma_i \leftarrow M_\Sigma(pp, \mathtt{pk}_b, \mathsf{Sign}(\mathtt{sk}_b, m_{(i,j)}), \Delta(s_i))$, and the correct offset $\Delta(s_i)$ derives from $C^*_{(i,2)}$. More specifically, $\mathcal{C}$ first obtains the randomness $\{r_j\}$ from updated dummy sketches, in which the dummy sketch is $\mathsf{SS}(s_j, r_j) = \mathbb{A}\cdot s_j + r_j$. Next, $\mathcal{C}$ can obtain the correct offset $\Delta(s_i)$ from a real sketch $\mathsf{SS}(s'_b, w_i) = \mathbb{A}\cdot\underline{X_b}+\chi+\mathbb{A}\cdot\Delta(s)$ and L dummy sketches, where the offset $\Delta(s)$ and the randomness $\{r_j\}$ are chosen at random by $\mathcal{A}$.
    In the g-th session of user i, upon receiving a Send query from $\mathcal{A}$, $\mathcal{C}$ first obtains $X'_b = X_b + \Delta(s_i)$, where $X_b = \underline{U}$ and the computation of correct offset $\Delta(s_i)$ using the same method described above; $\mathcal{C}$ then generates the re-randomized secret key $\mathtt{sk}'_b$ from X'_b (i.e., $\mathtt{sk}'_b \leftarrow \mathtt{H}_z(X'_b)$) for producing message-signature pair while $\mathcal{A}$ verifies it using the corresponding public key $\mathtt{pk}'_b$.
  * If $\mathcal{A}$ issues a Biometrics Reveal query to user i, then $\mathcal{C}$ aborts.
  * If $\mathcal{A}$ issues a Secret Key Reveal query to an instance oracle $\Pi^g_{\mathtt{pk}_i}$ (g-th session of user i), then $\mathcal{C}$ returns a (re-randomized) secret key $\mathtt{sk}'_b$ to $\mathcal{A}$. Notice that $\mathcal{A}$ is not allowed to obtain user i's enrolled secret key $\mathtt{sk}_b$.

In the Challenge session of user i, if the challenge of $\mathcal{C}$ is $\underline{X_{1,\cdots,k}}$, then the simulation is consistent with $\mathbb{G}_2$; Otherwise, the simulation is consistent with $\mathbb{G}_3$. If the advantage of $\mathcal{A}$ is significantly different in $\mathbb{G}_2$ and $\mathbb{G}_3$, then $\mathcal{C}$ can break the D-LWE$_{q,n_0-k,m_0,\chi}$. Since at most n users involved in the system, hence we have

$$\left|\mathtt{Adv}_2^{\text{pBRUA}} - \mathtt{Adv}_3^{\text{pBRUA}}\right| \leqslant n\cdot \mathtt{Adv}_{\mathcal{C}}^{\text{D-LWE}_{q,n_0-k,m_0,\chi}}(\lambda). \tag{3}$$

- $\mathbb{G}_4$: This game is identical to game $\mathbb{G}_3$ except that in the g-th session, the enrolled secret key $\mathtt{sk}_i$ is replaced by a random value u. Below we show that the difference between $\mathbb{G}_3$ and $\mathbb{G}_4$ is bounded by a negligible probability.
  Let $\mathcal{C}$ simulate the whole environment honestly according to the protocol specification, and it is easy to see that all the queries made to a user can be simulated perfectly using the user's secret keys and biometrics. In particular, the enrolled secret key of user i is $\mathtt{sk}_i$. In the g-th session of user i, to answer the Send query from $\mathcal{A}$, $\mathcal{C}$ will simulate the protocol transcript $T_i = (n_j, C^*_{(i,2)})$ as follows. $\mathcal{C}$ first simulates the real sketch as $\mathsf{SS}(s'_i, w_i) \leftarrow \mathbb{A}\cdot(\underline{s_i}+\Delta(s))+w_i$, where $s'_i = \underline{s_i}+\Delta(s)$, $\underline{s_i} \xleftarrow{\mathrm{R}} \mathtt{H}_z^{-1}(\underline{u}), u \in$

$\mathbb{Z}_q$, and $\Delta(s)$ is randomly chosen by $\mathcal{C}$; $\mathcal{C}$ then generates a secret/public key pair $(\mathtt{sk}'_i, \mathtt{pk}'_i)$ from a real sketch (with hidden secret s'_i) and L dummy sketches for producing the message-signature pair, where $\mathtt{sk}'_i \leftarrow \mathtt{H}_z(s'_i)$.
We then analyze the statistical distance between distribution $T_i = (n_j, C^*_{(i,2)})$ at game $\mathbb{G}_4$ and distribution T_i at previous game $\mathbb{G}_3$. We notice that the only difference is the simulated value $s_i \xleftarrow{\mathrm{R}} \mathtt{H}_z^{-1}(u)$ instead of taking the real enrolled secret key $\mathtt{sk}_i$ as input, and according to Lemma 2.2, we have the statistically distance between $\mathtt{sk}_i \leftarrow \mathtt{H}_z(s_i)$ and $u \in \mathbb{Z}_q$ with probability no greater than ϵ. Hence we have

$$\left|\mathtt{Adv}_3^{\mathrm{pBRUA}} - \mathtt{Adv}_4^{\mathrm{pBRUA}}\right| \leqslant \mathtt{Adv}_{\mathcal{C}}^{\mathbb{H}}(\lambda). \tag{4}$$

- $\mathbb{G}_5$: This game is identical to game $\mathbb{G}_4$ except that in the g-th session, $\mathcal{C}$ outputs a random bit if **Forge** event happens where $\mathcal{A}$'s Send query includes a valid forgery σ^* while user i's enrolled secret key is not corrupted. Then we have

$$\left|\mathtt{Adv}_4^{\mathrm{pBRUA}} - \mathtt{Adv}_5^{\mathrm{pBRUA}}\right| \leqslant \Pr[\textbf{Forge}]. \tag{5}$$

Let $\mathcal{F}$ denote a forger against a signature scheme Σ with EUF-CMA security, who is given a public key $\mathtt{pk}^*$ and a signing oracle $\mathcal{O}$, aims to find a forgery σ^*. $\mathcal{C}$ simulates the game for $\mathcal{A}$ as follows.

* Setup. $\mathcal{F}$ sets up the game for $\mathcal{A}$ by creating n users with the corresponding block identifiers and biometrics. $\mathcal{F}$ sets up user i's enrolled block as $(\mathtt{pk}^*, \mathsf{SS}(s_i, w_i))$, where sketch is $\mathsf{SS}(w_i, \mathtt{sk}_i) = \mathbb{A} \cdot s_i + w_i, s_i \in \mathbb{Z}_q^n$. $\mathcal{F}$ also honestly generates public/secret key pairs and sketches as Enrollment specified for n-1 users. Eventually, $\mathcal{F}$ sends all enrolled real/dummy public keys and sketches to $\mathcal{A}$. Note that $\mathcal{A}$ cannot link the simulated sketch $\mathsf{SS}(s_i, w_i)$ to the public key $\mathtt{pk}^*$ since $\mathcal{A}$ is not allowed to access user i's biometrics w_i.
* Training. $\mathcal{F}$ answers $\mathcal{A}$'s queries as follows.
  * If $\mathcal{A}$ issues a Send query in the form of $(n_j, C^*_{(i,2)})$ to $\mathcal{F}$, $\mathcal{F}$ chooses a response randomness n_i first, then $\mathcal{F}$ simulates the protocol transcript $T_i = (m_{(i,j)}, \sigma'_i)$ as follows
    - invoke the signing oracle $\mathcal{O}$ to obtain a message-signature pair $(m_{(i,j)}, \sigma_i)$, where $m_{(i,j)} = (n_i, n_j, \text{Request})$;
    - obtain the correct offset $\Delta(s_i)$ from $C^*_{(i,2)}$, where $C^*_{(i,2)}$ includes a (L+1 size) set of (re-randomized) real and dummy sketches. Note that the real sketch is $\mathsf{SS}(s'_i, w_i) \leftarrow \mathsf{SS}(s_i, w_i) + \mathbb{A} \cdot \Delta(s_i)$;
    - generate a signature $\sigma'_i \leftarrow M_\Sigma(pp, \mathtt{pk}^*, \sigma_i, \Delta(s_i))$ by using the deterministic algorithm M_Σ described in Section 2.2; Note that $\mathtt{sk}'_i \leftarrow \mathtt{H}(s'_i)$.
    - return $(m_{(i,j)}, \sigma'_i)$ to $\mathcal{A}$.
  * If $\mathcal{A}$ issues a Secret Key Reveal query to an instance oracle $\Pi^i_{\mathtt{pk}^*}$, then $\mathcal{F}$ returns a re-randomized secret key $\mathtt{sk}'_i$ to $\mathcal{A}$. Since $\mathcal{A}$ is not allowed to reveal the enrolled secret key $Dlog(\mathtt{pk}^*)$, the simulation is perfect.
* When **Forge** event occurs (i.e., $\mathcal{A}$ outputs $(m^*, \underline{\sigma^{*'}}, C^{*'}_{(i,2)})$, where $C^{*'}_{(i,2)} = \{\mathsf{SS}(s_i^{*'}, w_i)\}$), $\mathcal{F}$ checks whether:
  * the **Forge** event happens at g-th session;

$*$ the message-signature pair $(m^*, \underline{\sigma^{*'}})$ was not previously simulated by $\mathcal{C}$, where $m^* = (n_i, n_j, \text{Request})$;

$*$ verifies $\Sigma.\mathsf{Verify}(\mathtt{pk}^{*'}, m^*, \underline{\sigma^{*'}}) = 1$, where $\mathtt{pk}^{*'} \leftarrow \mathtt{pk}^* \cdot \mathsf{KG}'(pp, \Delta(\mathtt{sk}^*))$ and $\Delta(\mathtt{sk}^*)$ derives from $C^{*'}_{(i,2)}$ that includes a (L+1 size) set of (re-randomized) real/dummy sketches. Note that $\Delta(\mathtt{sk}^*)$ is the correct "offset" between $Dlog(\mathtt{pk}^{*'})$ and $Dlog(\mathtt{pk}^*)$.

If all the above conditions hold, $\mathcal{F}$ confirms that it as a successful forgery from $\mathcal{A}$, then $\mathcal{F}$ extracts the forgery via $\underline{\sigma^*} \leftarrow M_\Sigma(pp, \mathtt{pk}^*, \underline{\sigma^{*'}}, \Delta(\mathtt{sk}^*))$ using the homomorphic property of Σ. To this end, $\mathcal{F}$ outputs $\underline{\sigma^*}$ as its own forgery; Otherwise, $\mathcal{F}$ aborts the game. Therefore, we have

$$|\Pr[\mathbf{Forge}]| \leqslant \mathtt{Adv}^{\text{EUF-CMA}}_{\mathcal{F}}(\lambda). \tag{6}$$

It is easy to see that in game $\mathbb{G}_5$, $\mathcal{A}$ has no advantage, i.e.,

$$\mathtt{Adv}^{\text{pBRUA}}_{5} = 0. \tag{7}$$

Combining the above results together, we have

$$\mathtt{Adv}^{\text{pBRUA}}_{\mathcal{A}}(\lambda) \leqslant n \cdot m^2/2^\lambda + m[n \cdot \mathtt{Adv}^{\text{D-LWE}_{q,n_0-k,m_0,\chi}}_{\mathcal{C}}(\lambda) + \mathtt{Adv}^{\mathbb{H}}_{\mathcal{C}}(\lambda) + \mathtt{Adv}^{\text{EUF-CMA}}_{\mathcal{F}}(\lambda)].$$

□

Theorem 5.2. *The proposed pBRUA achieves strong privacy if the family of universal hash functions $\mathbb{H} \leftarrow \{\mathtt{H}_z : \mathbb{Z}_q^{n_0} \rightarrow \mathbb{Z}_q\}_{z \in Z}$ is ϵ-statistically secure, the underlying computational fuzzy extractor is IND secure, public key encryption scheme is IK-CPA secure, and the access pattern under uORAM protocol is computationally IND secure.*

Proof. We define a sequence of games $\{\mathbb{G}_i\}$ and let $\mathtt{Adv}^{\text{pBRUA}}_i$ denote the advantage of the adversary $\mathcal{A}$ in game $\mathbb{G}_i$. We also highlight the differences between adjacent games by underline. We assume the Challenge stage between adversary $\mathcal{A}$ and challenger $\mathcal{C}$ is executed in a specific time-window.

- $\mathbb{G}_0$: This is the original game for user anonymity security.
- $\mathbb{G}_1$: This game is identical to game $\mathbb{G}_1$ except that at the Challenge stage, we replace the real sketch $\mathsf{SS}(s_i, w_i)$ (i.e., p_i) by a random vector $r \in \mathbb{Z}_q^{m_0}$. Below we show the difference between $\mathbb{G}_0$ and $\mathbb{G}_1$ is negligible under the assumption that the computational fuzzy extractor (FE) is IND secure.
  Let $\mathcal{C}$ denote an attacker, who is given a common public matrix $\mathbb{A}$ and two sketches $(\mathtt{pk}_0, \mathtt{pk}_1)$ ($\mathtt{pk} \leftarrow \mathbb{A} \cdot X + \chi$), aims to break the IND security of the computational FE. $\mathcal{C}$ simulates the game for $\mathcal{A}$ as follows.

  $*$ Setup. $\mathcal{C}$ sets up the game for $\mathcal{A}$ by creating n users with corresponding block identifiers $\{ID_i\}$ and leaf identifiers $\{leaf_{ID_i}\}$. $\mathcal{C}$ sets the common public matrix in the system as $\mathbb{A}$. $\mathcal{C}$ randomly chooses users i, j from user set $\mathcal{U}$ and sets the enrolled sketches as $p_i = \mathtt{pk}_0, p_j = \mathtt{pk}_1$, and generates biometrics and sketches for other users. In addition, $\mathcal{C}$ honestly generates enrolled public/secret key pairs $\{\mathtt{pk}_i, \mathtt{sk}_i\}$ for n users.

  $*$ Training. If $\mathcal{A}$ issues a Send query in the form of $\{\mathtt{pk}_i, p_i\}$ (assuming the bucket includes a real block) to user i during **EarlyReshuffles**, then $\mathcal{C}$ performs the following steps

∗ re-randomize the received real/dummy blocks to $\{\mathtt{pk}'_i, p'_i\}$ by using the homomorphic property of public key and computational FE respectively; Note that p'_i is derived from p_i.

∗ reshuffle the updated blocks according to the random offset which is based on PRP;

∗ encrypt the ID_i, random offset and leaf identifier $leaf_{ID_i}$ under enrolled public key $\mathtt{pk}_i$, and generate the **Bucket MetaData**;

∗ write back to the specified bucket as **EarlyReshuffles** operation specified.

If $\mathcal{A}$ issues a Send query in the form of $(\{p'_i\}, n_j)$ (the set $\{p'_i\}$ includes one real sketch and the involving dummy sketches) to user i, then $\mathcal{C}$ performs the simulation as follows.

∗ choose the randomness n_i as response;

∗ simulate the message-signature pair $(m_{(i,j)}, \sigma^*_i)$ using user i's enrolled secret key and correct offset which derives from two re-randomized sketches $\{p_i\}$ and $\{p'_i\}$.

∗ perform the **Evict** operation.

Note that $\mathcal{C}$ can easily identify the real block by the enrolled public key $\mathtt{pk}_i$, and meanwhile, $\mathcal{C}$ records the re-randomized real public key $\mathtt{pk}'_i$. Similarly, $\mathcal{C}$ simulates the response of user j using the same method as above.

∗ Challenge. If $\mathcal{A}$ issues a Send query in the form of $\{\mathtt{pk}_i, p_i\}$ (include a re-randomized block of user i) to challenge user $\mathtt{pk}_b$ during **EarlyReshuffles**, then $\mathcal{C}$ updates (re-randomizes) the received real/dummy blocks to $\{\underline{(\mathtt{pk}'_i, c^{*'})}, (\mathtt{pk}_l, p_l)\}$ ($l \neq i$), where the re-randomized ciphertext $c^{*'}$ derives from a challenge ciphertext $\underline{c^*}$ due to the linearity property of computational FE. Note that $\underline{c^*}$ is the challenge ciphertext on message m^* (the message is the offset between two re-randomized sketches). $\mathcal{C}$ performs the simulation of $\mathtt{pk}_b$ during **Evict** operation using the same method described above. Similarly, $\mathcal{C}$ simulates the response of $\mathtt{pk}_b$ using the same method when $\mathcal{A}$ issues a Send query that includes a real block of user j.

Finally, $\mathcal{C}$ outputs whatever $\mathcal{A}$ outputs. If $\mathcal{A}$ guesses the random bit correctly, then $\mathcal{C}$ can break the IND security of computational FE. Since at most n users involved in the system, hence we have

$$\left|\mathtt{Adv}_0^{\mathrm{pBRUA}} - \mathtt{Adv}_1^{\mathrm{pBRUA}}\right| \leqslant n \cdot \mathtt{Adv}_{\mathcal{C}}^{\mathrm{FE}}(\lambda). \tag{8}$$

- $\mathbb{G}_2$: This game is identical to game $\mathbb{G}_1$ except that at the Challenge stage, we replace the real secret key $\mathtt{sk}_i$ by a random key $r \in \mathbb{Z}_q$. By following the same analysis as described in game $\mathbb{G}_4$ of previous proof, we have

$$\left|\mathtt{Adv}_1^{\mathrm{pBRUA}} - \mathtt{Adv}_2^{\mathrm{pBRUA}}\right| \leqslant \mathtt{Adv}_{\mathcal{C}}^{\mathbb{H}}(\lambda). \tag{9}$$

- $\mathbb{G}_3$: This game is identical to game $\mathbb{G}_2$ except that at the Challenge stage, we replace the real public key $\mathtt{pk}_i$ by a random key $r \in \mathbb{Z}_q$. Below we show the difference between $\mathbb{G}_2$ and $\mathbb{G}_3$ is negligible under the assumption that the public key encryption (PKE) scheme is IK-CPA secure.
  Let $\mathcal{C}$ denote an attacker, who is given two public key pair $(\mathtt{pk}_0, \mathtt{pk}_1)$, aims to break the IK-CPA security of the PKE. $\mathcal{C}$ simulates the game for $\mathcal{A}$ as follows.

  ∗ Setup. $\mathcal{C}$ sets up the game for $\mathcal{A}$ by creating n users with corresponding biometrics $\{w_i\}$ and sketches p_i (i.e., $\{\mathsf{SS}(s_i, w_i)\}$). $\mathcal{C}$ randomly chooses users i, j from user set $\mathcal{U}$ and sets $\mathtt{pk}_i = \mathtt{pk}_0, \mathtt{pk}_j = \mathtt{pk}_1$, and generates public/secret key pair for other users honestly. In addition, $\mathcal{C}$ honestly generates block identifiers $\{ID_i\}$ and leaf identifiers $\{leaf_{ID_i}\}$ for n users in the system.

$*$ Training. If $\mathcal{A}$ issues a Send query in the form of either $\{\mathtt{pk}_i, p_i\}$ or $(\{p_i'\}, n_j)$ to user i, then $\mathcal{C}$ performs the simulation by following the protocol specification honestly. In particular, $\mathcal{C}$ simulates the message-signature pair $(m_{(i,j)}, \sigma_i^*)$ using the method as described in [18], where $m_{(i,j)} = (n_i, n_j, \text{Request})$. Note that $\mathcal{C}$ can obtain the (one-time) public key $\mathtt{pk}_i^*$ ($\mathtt{pk}_i^*$ is derived from $\mathtt{pk}_i$) since the correct offset can be computed from the received real/dummy sketches. $\mathcal{C}$ simulates the response of user j using the same method.

$*$ Challenge. If $\mathcal{A}$ issues a Send query in the form of $\{\mathtt{pk}_i, p_i\}$ to challenge user $\mathtt{pk}_b$ during **EarlyReshuffles**, then $\mathcal{C}$ simulates the ciphertext stored in the **Bucket MetaData** as $\underline{c^*}$, where the challenge ciphertext $\underline{c^*}$ is on message m^* (e.g., $m^* \leftarrow (ID_i, offset, leaf_{ID_i})$) obtained from his challenger. $\mathcal{C}$ honestly performs the simulation of $\mathtt{pk}_b$ during **Evict** operation according to the protocol specification. Similarly, $\mathcal{C}$ can simulate the response of $\mathtt{pk}_b$ when $\mathcal{A}$ issues a Send query that includes a real block of user j.

Finally, $\mathcal{C}$ outputs whatever $\mathcal{A}$ outputs. If $\mathcal{A}$ guesses the random bit correctly, then $\mathcal{C}$ can break the IK-CPA security of PKE. Since at most log$N$ buckets involved during the **EarlyReshuffles** operation, we have

$$\left|\mathtt{Adv}_2^{\text{pBRUA}} - \mathtt{Adv}_3^{\text{pBRUA}}\right| \leqslant \log N \cdot \mathtt{Adv}_{\mathcal{C}}^{\text{PKE}}(\lambda). \tag{10}$$

- $\mathbb{G}_4$: This game is identical to game $\mathbb{G}_3$ except that at the Challenge stage, we replace the real block identifier ID_i by a random string. Below we show the difference between $\mathbb{G}_3$ and $\mathbb{G}_4$ is negligible under the assumption that the access patten under uORAM protocol is computationally IND secure.

  Let $\mathcal{C}$ denote an attacker, who is given two access patterns $(\overleftarrow{y_0}, \overleftarrow{y_1})$ with equal length in the time-window ϱ such that $\overleftarrow{y_0} = \{(op_i, ID_i, data_i, time_i)\}$, aims to break the IND security of uORAM protocol. $\mathcal{C}$ simulates the game for $\mathcal{A}$ as follows.

  $*$ Setup. $\mathcal{C}$ sets up the game for $\mathcal{A}$ by creating n users with corresponding biometrics $\{w_i\}$, sketches p_i (i.e., $\{\mathsf{SS}(s_i, w_i)\}$) and public/secret key pairs $\{(\mathtt{pk}_i, \mathtt{sk}_i)\}$. In addition, $\mathcal{C}$ randomly chooses users i, j from user set $\mathcal{U}$ and sets user i, j's access pattern as $(\overleftarrow{y_0}, \overleftarrow{y_1})$ respectively, and generates both block identifier and leaf identifier for other users honestly. Note that user i's block identifier is implicitly sets as $ID_i = ID_0$ with respect to access pattern $\overleftarrow{y_0}$ (assuming a user has one unique block identifier ID_0). Also note that $\mathcal{C}$ generates user i, j's leaf identifiers.

  $*$ Training. If $\mathcal{A}$ issues a Send query in the form of either $\{\mathtt{pk}_i, p_i\}$ or $(\{p_i'\}, n_j)$ to user i, then $\mathcal{C}$ simulates the response of user i by following the protocol execution honestly. In particular, $\mathcal{C}$ perfectly simulates the message-signature pair using user i's secret key and biometrics. $\mathcal{A}$ faithfully follows the access pattern $\overleftarrow{y_0}$ w.r.t. user i, the same rule applies to user j.

  $*$ Challenge. If $\mathcal{A}$ issues a Send query in the form of either $\{\mathtt{pk}_i, p_i\}$ or $(\{p_i'\}, n_j)$ to challenge user $\mathtt{pk}_b$, then $\mathcal{C}$ constructs an equal length access pattern $\underline{\overleftarrow{y_0^*}}$ which includes a random block identifier $\underline{r}$. Then $\mathcal{C}$ sends two access patterns to his challenge oracle and obtains a challenge sequence of operations under uORAM protocol $uORAM(\overleftarrow{y_b})$ and returns it to $\mathcal{A}$. In addition, $\mathcal{C}$ simulates the message-signature pair of $\mathtt{pk}_b$ using user i's secret key and correct offset which derives from a $(L+1$ size) set of real/dummy sketches that obtained from its challenger. Note that $\mathcal{C}$ can also simulate the response of $\mathtt{pk}_b$ when $\mathcal{A}$ issues a Send query that includes a real block of user j.

Finally, $\mathcal{C}$ outputs whatever $\mathcal{A}$ outputs. If $\mathcal{A}$ guesses the random bit correctly, then $\mathcal{C}$ can break the computationally IND security of uORAM. Hence we have

$$\left|\mathtt{Adv}_3^{\text{pBRUA}} - \mathtt{Adv}_4^{\text{pBRUA}}\right| \leqslant \mathtt{Adv}_{\mathcal{C}}^{\text{uORAM}}(\lambda). \tag{11}$$

It is easy to see that in game $\mathbb{G}_4$, $\mathcal{A}$ has no advantage, i.e.,

$$\mathtt{Adv}_4^{\text{pBRUA}} = 0. \tag{12}$$

Combining the above results together, we have

$$\mathtt{Adv}_{\mathcal{A}}^{\text{pBRUA}}(\lambda) \leqslant n \cdot \mathtt{Adv}_{\mathcal{C}}^{\text{FE}}(\lambda) + \mathtt{Adv}_{\mathcal{C}}^{\mathbb{H}}(\lambda) + \log N \cdot \mathtt{Adv}_{\mathcal{C}}^{\text{PKE}}(\lambda) + \mathtt{Adv}_{\mathcal{C}}^{\text{uORAM}}(\lambda).$$

□

6. Conclusion

In this work, we have proposed the first general framework of strong privacy-preserving remote user authentication based on a new uORAM protocol and computational fuzzy extractors. The proposed general framework achieves the strong privacy against an honest-but-curious authentication server. In particular, the general framework supports a constant bandwidth cost in the challenge-response phase of user authentications. We have proved the security of the proposed general framework under standard assumptions. As for the future work, we would try to design a strong privacy-preserving user authentication that 1) handles multiple user requests in a concurrent and asynchronous manner [29, 30]; or 2) secures against malicious servers [31, 44].

References

[1] X. Boyen, Reusable cryptographic fuzzy extractors, in: *ACM CCS*, 2004, pp. 82–91.

[2] X. Boyen, Y. Dodis, J. Katz, R. Ostrovsky and A.D. Smith, Secure Remote Authentication Using Biometric Data., in: *EUROCRYPT*, Vol. 3494, 2005, pp. 147–163.

[3] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk and T. Toft, Privacy-preserving face recognition, in: *PET*, 2009, pp. 235–253.

[4] M. Barni, T. Bianchi, D. Catalano, M. Di Raimondo, R. Donida Labati, P. Failla, D. Fiore, R. Lazzeretti, V. Piuri, F. Scotti et al., Privacy-preserving fingercode authentication, in: *Proceedings of the 12th ACM workshop on Multimedia and security*, 2010, pp. 231–240.

[5] Y. Huang, L. Malka, D. Evans and J. Katz, Efficient Privacy-Preserving Biometric Identification, in: *NDSS*, 2011.

[6] N. Li, F. Guo, Y. Mu, W. Susilo and S. Nepal, Fuzzy Extractors for Biometric Identification, in: *ICDCS*, 2017, pp. 667–677.

[7] M.S. Islam, M. Kuzu and M. Kantarcioglu, Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation., in: *NDSS*, 2012, p. 12.

[8] M. Maffei, G. Malavolta, M. Reinert and D. Schröder, Privacy and access control for outsourced personal records, in: *Security and Privacy (SP)*, 2015, pp. 341–358.

[9] Is Your Information on Mobile Health Apps Safe?.

[10] B. Fuller, X. Meng and L. Reyzin, Computational fuzzy extractors, in: *ASIACRYPT*, 2013, pp. 174–193.

[11] D. Apon, C. Cho, K. Eldefrawy and J. Katz, Efficient, reusable fuzzy extractors from LWE, in: *International Conference on Cyber Security Cryptography and Machine Learning*, 2017, pp. 1–18.

[12] Y. Wen and S. Liu, Robustly reusable fuzzy extractor from standard assumptions, in: *ASIACRYPT*, 2018, pp. 459–489.

[13] Y. Dodis, L. Reyzin and A. Smith, Fuzzy extractors: How to generate strong keys from biometrics and other noisy data, in: *EUROCRYPT*, 2004, pp. 523–540.
[14] D. Chaum and E. Van Heyst, Group signatures, in: *EUROCRYPT*, 1991, pp. 257–265.
[15] F. Zhang and K. Kim, ID-based blind signature and ring signature from pairings, in: *ASIACRYPT*, 2002, pp. 533–547.
[16] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan, Private information retrieval, in: *Proceedings of IEEE 36th Annual Foundations of Computer Science*, 1995, pp. 41–50.
[17] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi and P. Samarati, Shuffle index: Efficient and private access to outsourced data, *ACM Transactions on Storage (TOS)* **11**(4) (2015), 19.
[18] C.-P. Schnorr, Efficient identification and signatures for smart cards, in: *CRYPTO*, 1989, pp. 239–252.
[19] K. Takahashi, T. Matsuda, T. Murakami, G. Hanaoka and M. Nishigaki, A signature scheme with a fuzzy private key, in: *ACNS*, 2015, pp. 105–126.
[20] T. Matsuda, K. Takahashi, T. Murakami and G. Hanaoka, Fuzzy signatures: relaxing requirements and a new construction, in: *ACNS*, 2016, pp. 97–116.
[21] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu and S. Devadas, Path ORAM: an extremely simple oblivious RAM protocol, in: *ACM CCS*, 2013, pp. 299–310.
[22] L. Ren, C.W. Fletcher, A. Kwon, E. Stefanov, E. Shi, M. Van Dijk and S. Devadas, Constants Count: Practical Improvements to Oblivious RAM., in: *USENIX*, 2015, pp. 415–430.
[23] O. Goldreich and R. Ostrovsky, Software protection and simulation on oblivious RAMs, *JACM* **43**(3) (1996), 431–473.
[24] E. Shi, T.-H.H. Chan, E. Stefanov and M. Li, Oblivious RAM with O ((logN) 3) worst-case cost, in: *ASIACRYPT*, 2011, pp. 197–214.
[25] J.L. Dautrich Jr, E. Stefanov and E. Shi, Burst ORAM: Minimizing ORAM Response Times for Bursty Access Patterns., in: *USENIX*, 2014, pp. 749–764.
[26] E. Stefanov, E. Shi and D.X. Song, Towards Practical Oblivious RAM, in: *NDSS*, 2012.
[27] S. Devadas, M. van Dijk, C.W. Fletcher, L. Ren, E. Shi and D. Wichs, Onion ORAM: A constant bandwidth blowup oblivious RAM, in: *TCC*, 2016, pp. 145–174.
[28] X. Wang, H. Chan and E. Shi, Circuit ORAM: On tightness of the goldreich-ostrovsky lower bound, in: *ACM CCS*, 2015, pp. 850–861.
[29] E. Stefanov and E. Shi, Oblivistore: High performance oblivious cloud storage, in: *Security and Privacy (SP)*, 2013, pp. 253–267.
[30] V. Bindschaedler, M. Naveed, X. Pan, X. Wang and Y. Huang, Practicing oblivious access on cloud storage: the gap, the fallacy, and the new way forward, in: *ACM CCS*, 2015, pp. 837–849.
[31] C. Sahin, V. Zakhary, A. El Abbadi, H. Lin and S. Tessaro, Taostore: Overcoming asynchronicity in oblivious data storage, in: *Security and Privacy (SP)*, 2016, pp. 198–217.
[32] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz and D. Song, Phantom: Practical oblivious computation in a secure processor, in: *ACM CCS*, 2013, pp. 311–324.
[33] J. Doerner and A. Shelat, Scaling ORAM for secure computation, in: *ACM CCS*, 2017, pp. 523–535.
[34] A. Juels and M. Wattenberg, A fuzzy commitment scheme, in: *ACM CCS*, 1999, pp. 28–36.
[35] O. Regev, On lattices, learning with errors, random linear codes, and cryptography, *JACM* **56**(6) (2009), 34.
[36] T. Murakami, T. Ohki and K. Takahashi, Optimal sequential fusion for multibiometric cryptosystems, *Information Fusion* **32** (2016), 93–108.
[37] N. Döttling and J. Müller-Quade, Lossy codes and a new variant of the learning-with-errors problem, in: *EUROCRYPT*, 2013, pp. 18–34.
[38] A. Akavia, S. Goldwasser and V. Vaikuntanathan, Simultaneous hardcore bits and cryptography against memory attacks, in: *TCC*, 2009, pp. 474–495.
[39] J. Kamp and D. Zuckerman, Deterministic extractors for bit-fixing sources and exposure-resilient cryptography, *SIAM* **36**(5) (2006), 1231–1247.
[40] O. Blazy, G. Fuchsbauer, D. Pointcheval and D. Vergnaud, Signatures on randomizable ciphertexts, in: *PKC*, 2011, pp. 403–422.
[41] T. ElGamal, A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE transactions on information theory* **31**(4) (1985), 469–472.
[42] M. Bellare, A. Boldyreva, A. Desai and D. Pointcheval, Key-privacy in public-key encryption, in: *ASIACRYPT*, 2001, pp. 566–582.
[43] R. Cramer and V. Shoup, A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack, in: *CRYPTO*, 1998, pp. 13–25.
[44] M. Backes, A. Herzberg, A. Kate and I. Pryvalov, Anonymous RAM, in: *ESORICS*, 2016, pp. 344–362.

Appendix A. Trivial Solution

Suppose that we use a single instance of Path ORAM as a black-box with N records of n users ($N \geqslant n$), and we show that we can achieve strong privacy and log-linear time-complexity, *not* constant bandwidth. Specifically, we let the authentication server construct a binary tree, in which each leaf/non-leaf node is a bucket, each bucket contains a certain number of records and each record contains a user's enrolled verification key and helper data (or sketch). Note that helper data and a cryptographic key are derived from a user's biometrics, and the same key can be extracted by inputting a "nearby" biometrics and the helper data [13]. In particular, the cryptographic key is used to derive a signing/verification key pair. The idea is, upon an authentication request, the authentication server retrieves a set of records which reside in the same tree path from the database and returns them back to the user. The authorized user first obtains a cryptographic key from her "nearby" biometrics and her enrolled helper data that are included in the returned helper data set, and then generates an anonymous signature (e.g., group signature [14] or ring signature [15]) using a signing key derived from the cryptographic key. The authentication server verifies the anonymous signature under a number of verification keys that are involved in the returned records set. This solution naturally achieves the log-linear time-complexity due to the structure of a binary tree. However, the bandwidth overhead has the log-linear time-complexity in the number of records N.

To highlight the advantage of uORAM over Path/Ring ORAM in user authentications, we present the following points. First, Path ORAM protocol cannot support a constant bandwidth because the server simply returns all blocks in a tree path to the client. In uORAM (and Path ORAM), the server returns only one block from each bucket on the path, so that eliminating the dependence on the bucket size. Second, Path ORAM cannot support an **EarlyReshuffles** procedure. The uORAM utilizes an **EarlyReshuffles** procedure (same as Ring ORAM) to reshuffle the buckets, because the dummy blocks may have been updated too many times. In other words, the user must perform an **EarlyReshuffles** procedure before actual user authentication, in order to ensure the unlinkability across multiple authentications. Third, Ring ORAM is not suitable for user authentications, due to its **Evict** procedure (as discussed in the Related Work). Therefore, we must select the ideas from both Path ORAM and Ring ORAM to construct our uORAM, which should be particularly suitable for user authentications. We stress that the constant bandwidth in the challenge-response phase can be achieved if and only if we exploit the LWE-based fuzzy extractor. The benefit is to let the user authenticate herself to the server in a practical manner. In other words, the user can perform a fast login during challenge-response phase, while the early-reshuffle and post-reshuffle phases are used to maintain strong privacy. Lastly, the overall bandwidth in the whole user authentication can be summarized to read and write all real blocks (also includes some "updated" dummy blocks) in a tree path twice.